НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «*Засоби автоматизованого тестування інтерфейсу користувача*»

Виконала: студентка *IV* курсу, групи *ІС-62*

Путова Юлія Андріївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., Ліщук Катерина Ігорівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Новінський Валерій Петрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц. Ткач Михайло Мартинович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Путовій Юлії Андріївні
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Засоби автоматизованого тестування інтерфейсу користувача.*»
керівник проєкту *Ліщук Катерина Ігорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна варіантів використань

2. Схема структурна активності системи

3. Схема бази даних

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності програмного забезпечення

6. Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*15.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*18.04.2020*	
3.	*Постановка та формалізація задачі*	*20.04.2020*	
4.	*Розробка інформаційного забезпечення*	*24.04.2020*	
5.	*Алгоритмізація задачі*	*01.05.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*05.05.2020*	
7.	*Розробка програмного забезпечення*	*10.05.2020*	
8.	*Налагодження програми*	*13.05.2020*	
9.	*Виконання графічних документів*	*19.05.2020*	
10.	*Оформлення пояснювальної записки*	*25.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Юлія ПУТОВА

Керівник Катерина ЛІЩУК

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6218.00.000 ПЗ	Пояснювальна записка	103	
3	А4	ДП 6218.01.000 ТЗ	Технічне завдання	9	
4	А4	ДП 6218.02.000 ССД	Схема структурна діяльності	1	
5	А4	ДП 6218.03.000 ССВ	Схема структурна варіантів використань	1	
6	А4	ДП 6218.04.000 СБД	Схема бази даних	1	
7	А4	ДП 6218.05.000 ССК	Схема структурна компонентів	1	
			програмного забезпечення		
8	А3	ДП 6218.06.000 ССК	Схема структурна класів програмного	1	
			забезпечення		
9	А3	ДП 6218.07.000 ССП	Схема структурна послідовності	1	
10	А3	ДП 6218.08.000 КЕ	Креслення вигляду екранних форм	1	
11	А3	ДП 6218.09.000 КЗ	Креслення вигляду звітних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6218 00.000.00		
Розробн.		Путова Ю.А.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Ліщук К.І.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-62	
Н/контр.		Новінський В.П.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «*Засоби автоматизованого тестування інтерфейсу користувача*»

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з шести розділів, містить 24 рисунків, 23 таблиць, 1 додатків, 14 джерел.

Дипломний проєкт присвячений розробці комплексу задач автоматизованого тестування інтерфейсу користувача. Ціль розробки – створити фреймворк, що полегшує роботу автоматизаторів тестування за рахунок зменшення часу на розробку та виконання тестів, підвищує ефективність тестів.

Задачі розробки: ведення сценаріїв тестування, ведення очікуваних результатів для сценаріїв тестування, виконання тестових сценаріїв, ведення результатів виконання сценаріїв тестування, формування звітних форм.

У розділі інформаційного забезпечення описано вхідні та вихідні дані, структуру бази даних.

Розділ математичного забезпечення присвячений змістовній та математичній постановкам задачі, обґрунтуванню вибору алгоритму для розв’язання задачі.

У розділі програмного забезпечення описано засоби розробки програмного забезпечення, наведено діаграми класів, послідовності, компонентів.

У технологічному розділі наведено інструкцію користувача та результати випробування.

АВТОМАТИЗАЦІЯ, ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ, ПОРІВНЯННЯ ЗОБРАЖЕНЬ, ПЕРЦЕПТИВНИЙ ХЕШ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of six sections, contains 24 figures, 23 tables, 1 appendices, 14 sources.

The diploma project is devoted to the development of a set of tasks for automated testing of the user interface. The purpose of development is to create a framework that facilitates the work of test automation engineers by reducing the time for development and execution of tests, increases the efficiency of tests.

Development tasks: maintenance of testing scenarios, maintenance of expected results for testing scenarios, execution of test scenarios, maintenance of results of execution of testing scenarios, generation of reporting forms.

The information support section describes the input and output data, the structure of the database.

The section of mathematical support is devoted to the meaningful and mathematical formulations of the problem, substantiation of the choice of algorithm for solving the problem.

The software section describes the tools for software development, diagrams of classes, sequences, components.

The technological section provides user instructions and test results.

AUTOMATION, FUNCTIONAL TESTING,
COMPARISON OF IMAGES, PERCEPTIVE HASH.

ЗМІСТ

ВСТУП

Тестування програмного забезпечення – це процес аналізу програмного продукту та його документації з метою покращення якості програмного продукту. Тестування дозволяє впевнитись, що програмний продукт відповідає вимогам замовника та визначає умови, при яких робота програми є некоректною.

Одна з задач тестування графічного інтерфейсу – підтвердити, що застосування відповідає макету прототипу. Це перевірка елементів дизайну, таких як колір, шрифт, його розмір, розташування елементів один відносно одного. Таке тестування проводиться після додавання нового функціоналу до будь-якого компоненту або зміни дизайну застосування. Постійне тестування інтерфейсу користувача допомагає виявити дефекти у новому функціоналі або дефекти, що викликані цими змінами.

Головною задачею автоматизації тестування є зменшення часу та зусиль, що потребує ручне тестування. Автоматизовані функціональні тести у більшості випадків не можуть виявити дефекти у зовнішньому вигляді елементів сторінки та оцінити зовнішній вигляд сторінки у цілому. Одне з рішень цієї проблеми – це перевірка зовнішнього вигляду за допомогою порівняння знімків екрана сторінок або їх елементів. При цьому зміна розміру зображень або їх якості (як, наприклад при використанні алгоритмів стиснення зображень з втратою якості) не повинна впливати на результат порівняння, адже це може зробити тест нестабільним та погіршити якість тестування, так як один і той самий тест може виконуватися машинах з різними дисплеями. Іншою проблемою є те, що у різних браузерах шрифт тексту та кольори вебелементів можуть відрізнятися, хоча ці зміни можуть бути не помітними для людського ока.

Дипломний проєкт присвячений розробці комплексу задач автоматизації тестування інтерфейсу користувача.

Практичне значення одержаних результатів. Розроблено фреймворк для виконання функціонального тестування веб-застосувань. Розроблено алгоритм порівняння зображень.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Тестування програмного забезпечення – це процес аналізу програмного продукту та його документації з метою покращення якості програмного продукту. Тестування дозволяє впевнитись, що програмний продукт відповідає вимогам замовника та визначає умови, при яких робота програми є некоректною.

Об’єктом автоматизації є процес тестування графічного інтерфейсу користувача вебсторінок. На етапі тестування інтерфейсу користувача (UI) перевіряється, наскільки він зручний для користувача та чи відповідає він вимогам замовника. При цьому проводиться перевірка того, як відображаються елементи інтерфейсу при виконанні різних дій користувачем.

Одна з задач тестування графічного інтерфейсу – підтвердити, що застосування відповідає макету прототипу. Це перевірка елементів дизайну, таких як колір, шрифт, його розмір, розташування елементів один відносно одного. Таке тестування проводиться після додавання нового функціоналу до будь-якого компоненту або зміни дизайну застосування. Постійне тестування інтерфейсу користувача допомагає виявити дефекти у новому функціоналі або дефекти, що викликані цими змінами.

Тестування UI вебсайтів зазвичай проводиться на різних пристроях та браузерах. Це зумовлено тим, що при різних розмірах екрану пристроїв та у різних браузерах робота та зовнішній вигляд застосуванняків може відрізнятися. Як правило, вибір комбінацій браузерів, операційних систем та розмірів екрану є задачею замовника.

Головною задачею автоматизації тестування є зменшення часу та зусиль, що потребує ручне тестування. Виділяють наступні рівні

автоматизованого тестування: модульні тести (або unit-тести), що тестують окремі модулі програми (наприклад, один метод класу), інтеграційні тести, що призначені для перевірки того, як взаємодіють окремі модулі програми, та тестування інтерфейсу користувача. Запропонована Майком Коном піраміда тестування [1] (рисунок 1.1) показує, яка кількість тестів одного рівня по відношенню до інших є найбільш ефективною.

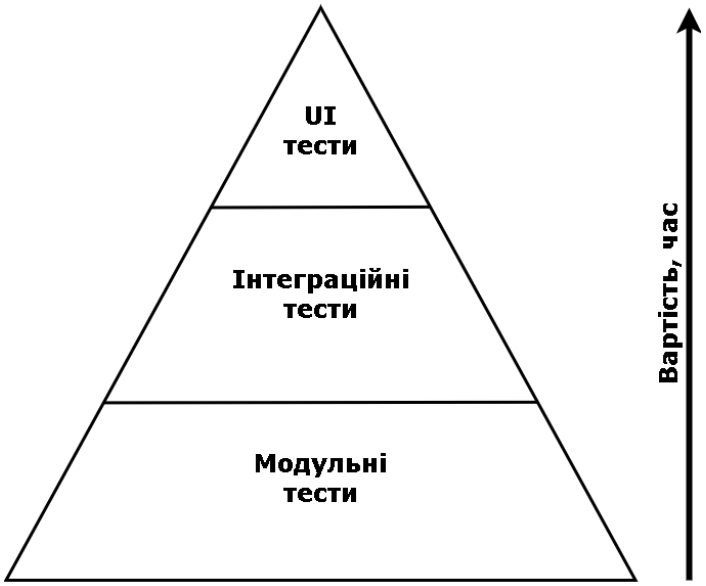


Рисунок 1.1 – Піраміда тестування.

Справді, для підготовки, виконання та підтримки тестів інтерфейсу користувача, потрібно значно більше часу, а отже і грошей замовника, ніж для розробки і виконання модульних тестів. Таким чином, доцільно мінімізувати кількість та тривалість UI тестів.

Найбільш популярними інструментами для автоматизованого тестування інтерфейсу користувача є Selenium, Unified Functional Testing (UFT), Katalon Studio, TestComplete. Selenium WebDriver – це бібліотеки, що дозволяють керувати браузерами. В них входять браузери Google Chrome, Firefox, Internet Explorer та Safari. Selenium є безкоштовним та підтримується операційними системами Windows, Linux та OS X. Бібліотеки Selenium

розроблені для мов програмування Java, .Net (C#), Python, JavaScript, та Ruby [2].

1.1.1 Опис процесу діяльності

Процес тестування інтерфейсу користувача передбачає аналіз вимог, написання сценаріїв, їх тестування та виконання, аналіз результатів виконання тестів. На рисунку 1.2 діаграма активностей показує основні етапи процесу тестування.

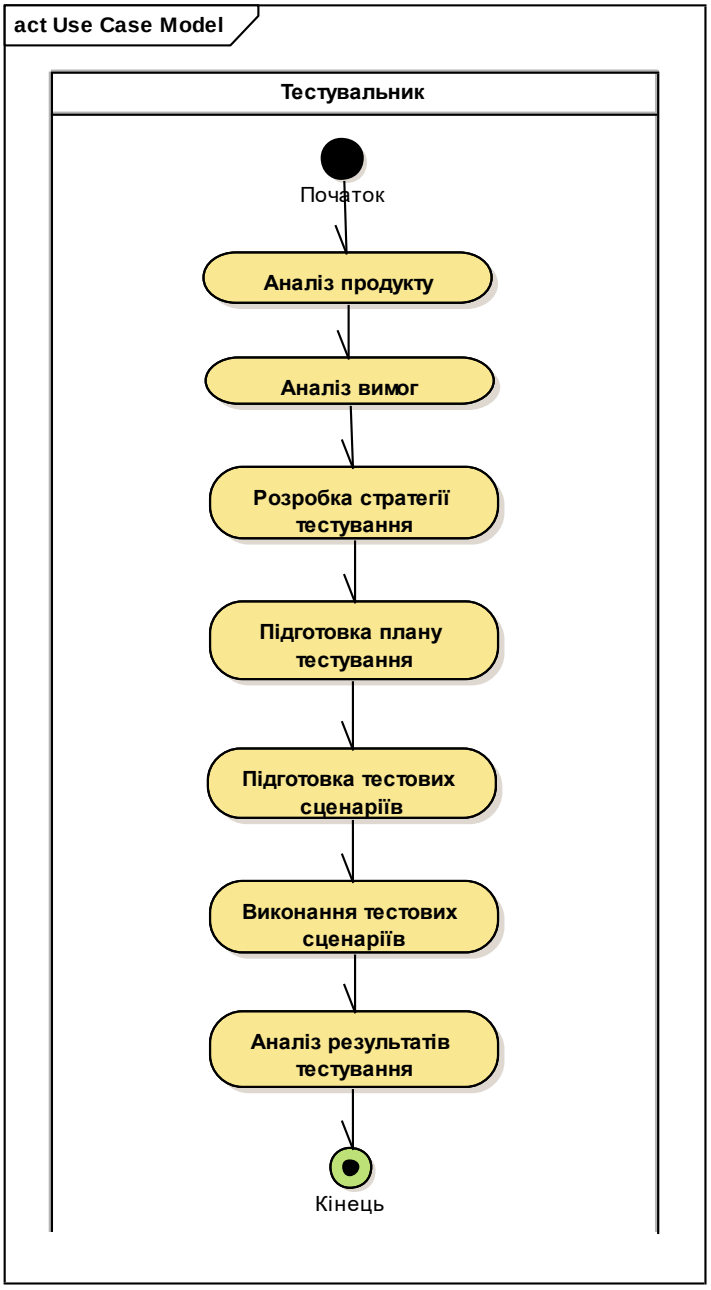


Рисунок 1.2 – Діаграма активностей.

При ручному тестуванні інтерфейсу користувача тестувальник самостійно перевіряє зовнішній вигляд елементів вебсторінок та вебсторінки у цілому. Автоматизоване тестування графічного інтерфейсу зазвичай передбачає перевірку вмісту атрибутів тегів, з яких складається вебелемент.

1.1.2 Опис функціональної моделі

У системі, що розробляється, актор – тестувальник програмного забезпечення – отримує інструменти для написання сценаріїв та їх імплементації з використанням методів тестування скріншотами, запуску тестів на різних браузерах та з різними розмірами екрану, генерації звіту виконаних тестів у формі HTML-сторінок. Розробка тестів відбувається за BDT (Behavior Driven Testing) методологією. BDT дозволяє не тільки тестувальникам, а й іншим членам команди брати участь у процесі створення тестових сценаріїв. У BDT тест-кейси (тестові сценарії) є абстрактними, написані природною мовою [3], а тому їх легко зрозуміти, відредагувати або навіть написати іншим членам команди з розробки ПЗ.

У таблиці 1.1 описані актори, варіанти використання та їх описи дій.

Таблиця 1.1 – Варіанти використання

Варіант використання	Опис дії варіанта використання	Актор
1 Написати тестові сценарії	Користувач може написати тестові сценарії природною мовою.	Тестувальник, член команди розробників
1.2 Протестувати тестові сценарії	Користувач може перевірити написані сценарії на відповідність до вимог замовника.	Тестувальник, член команди розробників

Продовження таблиці 1.1

1.3 Редагувати тестові сценарії	Користувач може редагувати тестові сценарії.	Тестувальник, член команди розробників
2 Розробити імплементацію кроків сценарію	Користувач може розробити імплементацію кроків сценарію для їх виконання.	Тестувальник
2.1 Завантажити зображення очікуваного результату	Користувач може завантажити зображення очікуваного вигляду вебсторінки або вебелемента.	Тестувальник
2.2 Отримати скріншот	Користувач може отримати скріншот вебсторінки або вебелемента при виконанні тестів.	Тестувальник
2.2.1 Отримати скріншот вебсторінки	Користувач може отримати скріншот вебсторінки при виконанні тестів.	Тестувальник
2.2.2 Отримати скріншот вебелемента	Користувач може отримати скріншот окремого вебелемента сторінки при виконанні тестів.	Тестувальник
2.3 Порівняти зображення	Користувач може отримати результат порівняння зображень при виконанні тестів.	Тестувальник
2.3.1 Означити елементи, що ігноруються при порівнянні	Користувач може вказати на вебелемент сторінки, зображення якого буде ігноруватися при порівнянні зображень.	Тестувальник

Продовження таблиці 1.1

2.3.2 Отримати візуалізацію результату порівняння	Користувач може переглядати результат порівняння зображень – зображення з виділеними кольором розбіжностями.	Тестувальник
3 Запустити виконання тестів	Користувач може запустити виконання окремих тестів або списку тестів.	Тестувальник
3.1 Обрати браузер для виконання тестів	Користувач може обрати браузер, на якому будуть виконуватися тести.	Тестувальник
3.2 Обрати розмір екрану для виконання тестів	Користувач може обрати розмір екрану, з яким буде відбуватися тестування.	Тестувальник
4 Отримати звіт з прогону тестів	Користувач може отримати звіт з прогону тестів у вигляді вебсторінки. Звіт має містити дані про розмір екрану та браузер, на якому виконувалися тести, назви виконуваних тестів, час виконання тестів, результати виконання тестів, зображення очікуваного та отриманого результатів, зображення з візуалізацією розбіжностей.	Тестувальник, член команди розробників.

Отже, діаграма прецедентів розроблюваної системи має наступний вигляд (рисунок 1.3):

Рисунок 1.3 – Діаграма прецедентів

Відповідно визначених варіантів використання у таблиці 1.2 наведені функціональні вимоги та встановлена їх пріоритетність.

Таблиця 1.2 – Функціональні вимоги

Функціональна вимога	Пріоритет
FR 1. Система надає можливість користувачу створювати автоматизовані тести за Behavior Driven Testing методологією.	Високий
FR 1.1 Система надає можливість користувачу описувати Given, When та Then кроки тестового сценарію латинськими літерами.	Високий
FR 1.2 Система надає можливість користувачу описувати передумови та післяумови для виконання тестового сценарію латинськими літерами.	Високий
FR 2. Система надає можливість користувачу редагувати тестові сценарії.	Високий
FR 3. Система надає можливість тестувальнику розроблювати імплементацію кроків тестового сценарію, передумов та післяумов сценарію.	Високий
FR 4. Система надає можливість тестувальнику ведення зображень очікуваних результатів.	Високий
FR 4.1 Система надає можливість тестувальнику додавати зображення очікуваних результатів.	Високий
FR 4.2 Система надає можливість тестувальнику видаляти зображення очікуваних результатів.	Високий
FR 5. Система надає можливість тестувальнику отримувати скріншот при виконанні тестів.	Високий
FR 5.1 Система зберігає отриманий скріншот.	Високий
FR 5.2 Система надає можливість тестувальнику отримувати скріншот вебсторінки при виконанні тестів.	Високий

Продовження таблиці 1.2

FR 5.3 Система надає можливість тестувальнику отримувати скріншот вебелемента при виконанні тестів.	Високий
FR 6. Система надає можливість тестувальнику виконувати порівняння зображень.	Високий
FR 6.1 Система надає можливість тестувальнику отримувати результат порівняння зображень.	Високий
FR 6.2 Система зберігає візуалізацію порівняння зображень.	Високий
FR 6.3 Система надає можливість порівнювати зображення, ігноруючи вебелементи.	Середній
FR 6.3.1 Система надає можливість тестувальнику означити вебелементи, зображення яких ігнорується при порівнянні.	Середній
FR 7 Система надає можливість тестувальнику запустити виконання тестів.	Високий
FR 7.1 Система запускає тести на обраному тестувальником браузері.	Середній
FR 7.1.1 Система надає можливість тестувальнику обрати браузер для виконання тестів.	Середній
FR 7.2 Система запускає тести з обраним тестувальником розміром екрану браузера.	Середній
FR 7.2.1 Система надає можливість тестувальнику ввести розмір екрану браузера для виконання тестів.	Середній
FR 8 Система надає можливість користувачу переглядати звіт з прогону тестів.	Високий

1.2 Огляд наявних аналогів

Тестування інтерфейсу користувача за допомогою знімків екрану вже використовується командами інженерів-автоматизаторів.

Однією з відомих бібліотек для створення скріншотів та для порівняння отриманих зображень є aShot. aShot – це бібліотека Java, створена командою Yandex [4]. aShot надає можливість робити скріншоти вебсторінок та їх окремих елементів на різних платформах (наприклад, може бути використаний при виконанні тестів на браузерах персональних комп'ютерів, симуляторах мобільних браузерів iOS та Android), виконувати обробку отриманих зображень, надає гнучкий інструмент для порівняння зображень.

AShot дозволяє порівнювати зображення, «ігнорувати» елементи або вказаний користувачем колір при порівнянні зображень, отримувати результати порівняння – відсоток розбіжності зображень та зображення з виділеними кольором пікселями, що не співпадають на порівнюваних скріншотах, що допомагає швидко знайти розбіжності людині, що переглядає результат.

Аналогічним інструментом є flue2ent. flue2ent – це API для створення наскрізних (end-to-end) UI тестів на Java [5]. flue2ent використовується разом з Selenium API та дозволяє полегшити реалізацію тестів і зробити код більш зрозумілим для сприйняття. Однією з його функцій є створення знімків екрану, редагування та порівняння зображень.

У випадку, коли на проекті використовується концепція неперервної інтеграції, для тестування зовнішнього вигляду вебелементів може бути використаний Happo – інструмент для крос-браузерного так крос-платформного тестування інтерфейсу користувача. Він використовує браузери Chrome, Firefox, Safari, iOS Safari, Edge та Internet Explorer, щоб робити скріншоти компонентів вебсторінок.

Коли розробник змінює код застосування і додає ці зміни до репозиторію, на виділеному сервері виконується автоматизоване складання проекту та виконання тестів. Happo зберігає знімки вебелементів до та після зміни коду застосування, порівнює їх, та повідомляє розробника про неспівпадіння, дозволяючи зрозуміти, що саме змінилося та чи очікувані ці зміни. Happo може працювати з основними службами неперервної інтеграції, таких як Travis CI, Jenkins та CircleCI [6].

ImageMagick – це пакет програм для роботи з графічними файлами, що може бути використаний як самостійний продукт, так і з мовами Perl, C, C++, C#, Python, Ruby, PHP, Java. ImageMagick може працювати більш, ніж з 200 форматами зображень у режимі читання та читання-запис [7]. До його функцій входить перетворення формату зображень, зміна розміру зображень, поворот або обрізка, нанесення рівномірного масштабування, додавання художніх ефектів, що включають зміну різкості або відтінків, додавання розмиття та багато інших. Крім цього, ImageMagick надає інструменти для порівняння зображень, додавання тексту або форм до зображень, накладання зображень, підтримує роботу з зображеннями великого розміру, анімаціями.

Для генерації звіту по візуальним тестам часто використовується Allure. Allure – це інструмент для створення звітів у вигляді вебсторінок. Він може бути інтегрований у тестові фреймворки на мовах програмування Java, Python, JavaScript, Ruby, Groovy, PHP, .Net, та Scala [8]. Звіт може містити у собі назви та параметри тестів, зображення очікуваного та фактичного результатів, результати тестів – відсоток схожості зображень та візуалізацію розбіжностей, вміст лог-файлів та ін.

1.3 Постановка задачі

1.3.1 Призначення розробки

Розроблювана система призначена для розробки автоматизованих функціональних тестів інтерфейсу користувача та їх виконання. Такі тести є ефективними для тестування веб-застосувань, де зміни в інтерфейсі користувача можуть призвести до втрати коштів замовника, та для таких, де ручне тестування інтерфейсу користувача є менш ефективним та більш часозатратним, ніж пропонований підхід.

1.3.2 Цілі та задачі розробки

Ціль розробки – створити фреймворк, що полегшує роботу автоматизаторів тестування за рахунок зменшення часу на розробку та виконання тестів, підвищує ефективність тестів, збільшуючи ймовірність знаходження дефектів (адже при ручному тестуванні дефекти можуть бути просто не помічені), розширює покриття застосування тестами та містить необхідні гнучкі інструменти для ефективного тестування інтерфейсу користувача та виведення результатів тестування у зручному вигляді.

Виконання тестів на різних браузерах та з різними розмірами екрану збільшує покриття програмного продукту тестами, не переписуючи тест кожного разу, що дозволяє якнайшвидше отримати дані про якість продукту.

Задачі розробки:

а) ведення сценаріїв тестування;

б) ведення очікуваних результатів для сценаріїв тестування;

в) виконання тестових сценаріїв;

г) ведення результатів виконання сценаріїв тестування;

д) формування звітних форм.

Висновок до розділу

У цьому розділі описано предметне середовище, розглянуто наявні аналоги розробки, описано призначення розробки, обґрунтовано цілі та задачі розробки. Розділ містить відомості про дійових осіб, які будуть працювати в межах розроблюваного програмного продукту та функції, які будуть виконуватися ними.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідні дані представлені у таблиці 2.1.

Таблиця 2.1 – Вхідні дані

Дані	Вміст даних
Тестові сценарії	Набір тестових сценаріїв, що мають структуру “Given-When-Then” та можливий опис передумов та післяумов для сценаріїв. Подається у файлах з розширенням .feature.
URL ресурсу	Текстовий рядок з адресою ресурсу, що тестується.
Локатори вебелементів	Текстовий рядок з адресою вебелемента на сторінці. Може подаватися у вигляді CSS-локатора, xPath або вмісту одного з атрибутів елемента: name, class, id.
Зображення очікуваних результатів	Растрове зображення очікуваного результату вигляду вебсторінки або вебелемента для окремого сценарія.
Реалізація кроків сценаріїв	Файл з програмним кодом реалізації кроків тестових сценаріїв.
Розмір екрану для виконання тестів	Текстовий рядок
Назва браузера для виконання тестів	Одне з значень DriverTypes - перелічуваного типу даних, що зберігається у файлі DriverTypes.cs

2.2 Вихідні дані

Вихідними даними є звіт з результатами виконання тестів у вигляді HTML-сторінки.

Звіт складається з наступних елементів: заголовок звіту, загальні результати виконання набору тестів, деталізований опис результату виконання кожного тесту.

Заголовок звіту містить дату та час генерації звіту.

Опис загальних результатів виконання набору тестів включає таблицю з наступними відомостями: кількість виконаних тестів, кількість успішно виконаних тестів, кількість пропущених тестів. Також показується відсоток успішності виконання тестів – відношення кількості успішно виконаних тестів до загальної кількості тестів та час, витрачений на виконання тестів, у форматі гг:хх:сс.чч, де компонент гг - години, вимірювані в 24-годинному форматі, хх - хвилини, сс - секунди, а чч - частки секунди.

Детальний опис кожного виконаного тесту приводиться у порядку їх виконання. Опис тесту включає назву набору тестових сценаріїв, до якого входить тест, назву тесту, результат виконання – статус завершення тесту та повідомлення про помилку, час виконання тесту у форматі гг:хх:сс.чч, де компонент гг - години, вимірювані в 24-годинному форматі, хх - хвилини, сс - секунди, а чч - частки секунди. Також приводяться скріншот очікуваного результату тесту, скріншот, отриманий при виконанні тесту та зображення, що є результатом порівняння цих скріншотів.

2.3 Опис структури бази даних

В базі даних SQL сформовано таблиці «Features», «Scenarios», «ResultStatuses», «Browsers», «Screens», «ExecutedTests» призначення яких – зберігати інформацію про тестові сценарії та набори тестових сценаріїв, конфігурацію, з якою виконуються тести (розміри екрану та назви браузерів), час виконання тестів, результати виконання тестів для подальшого збору статистик та аналізу.

У таблиці «Features» зберігаються назви наборів тестових сценаріїв (таблиця 2.2).

Таблиця 2.2 – Специфікація таблиці «Features»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
FeatureName	Назва набору тестових сценаріїв	VARCHAR	ні	ні

У таблиці «Scenarios» зберігаються назви тестових сценаріїв та ідентифікатори наборів, до яких вони належать (таблиця 2.3).

Таблиця 2.3 – Специфікація таблиці «Scenarios»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
ScenarioName	Назва тестового сценарію	VARCHAR	ні	ні
FeatureId	Id набору тестових сценаріїв	NUMBER	ні	ні

Таблиця «ResultStatuses» містить можливі статуси завершення тесту (таблиця 2.4).

Таблиця 2.4 - Специфікація таблиці «ResultStatuses»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
TestStatus	Назва стану завершення тесту	VARCHAR	ні	ні

У таблиці «Browsers» містяться назви вебпереглядачів, на яких проводиться тестування (таблиця 2.5).

Таблиця 2.5 – Специфікація таблиці «Browsers»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
BrowserName	Назва вебпереглядача	VARCHAR	ні	ні

У таблиці «Screens» зберігаються значення розмірів екрану, на яких проводиться тестування (таблиця 2.6).

Таблиця 2.6 – Специфікація таблиці «Screens»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
Size	Значення розміру екрану	VARCHAR	ні	ні

Таблиця «ExecutedTests» зберігає час початку і завершення виконання тестів, повідомлення про помилку (таблиця 2.7).

Таблиця 2.7 – Специфікація таблиці «ExecutedTests»

Поле	Опис	Тип даних	Первинний ключ	Може приймати значення null
Id	Ідентифікатор	NUMBER	так	ні
ScenarioId	Id тестового сценарію	NUMBER	ні	ні
BrowserId	Id браузера	NUMBER	ні	ні

Продовження таблиці 2.7

ScreenSizeId	Id розміру екрана	NUMBER	ні	ні
TestStatusId	Id статусу завершення виконання тесту	NUMBER	ні	ні
StartTime	Дата і час початку виконання тестового сценарію	DATETIME	ні	ні
EndTime	Дата і час завершення виконання тестового сценарію	DATETIME	ні	ні
ErrorMessage	Повідомлення про помилку	VARCHAR	Ні	так

Entity Relationship модель представлена на рисунку 2.1.

Рисунок 2.1 – ER-модель

Висновок до розділу

У цьому розділі описано вхідні та вихідні дані, структуру даних системи.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Автоматизовані функціональні тести у більшості випадків не можуть виявити дефекти у зовнішньому вигляді елементів вебсторінки та оцінити зовнішній вигляд вебсторінки у цілому. Одне з рішень цієї проблеми – це перевірка зовнішнього вигляду за допомогою порівняння скріншотів вебсторінок або їх елементів. При цьому зміна розміру зображень або їх якості (як, наприклад при використанні алгоритмів стиснення зображень з втратою якості) не повинна впливати на результат порівняння, адже це може зробити тест нестабільним та погіршити якість тестування, так як один і той самий тест може виконуватися машинах з різними дисплеями. Іншою проблемою є те, що у різних браузерах шрифт тексту та кольори вебелементів можуть відрізнятися, хоча ці зміни можуть бути не помітними для людського ока.

Побайтове порівняння зображень чутливе до таких змін, крім того, такий спосіб дуже ресурсозатратний, хоча і є простим у реалізації. Тому потрібно знайти такий метод порівняння зображень, який буде мати невеликий час виконання та показувати результати, що є допустимими для використання цього методу у автоматизованому тестуванні.

3.2 Математична постановка задачі

Знімок екрана – це растрове зображення, що є масивом точок (пікселів), що мають свій колір. Виконавши попередню обробку пікселів, як, наприклад, зменшення розміру зображення та переведення його кольорів в градації сірого, можна отримати дані, що характеризують зображення та дозволяють виконувати порівняння зображень, порівнюючи ці набори даних.

Зображення – це неперервна функція кольору від координат $c(x, y)$. При переведенні зображення до цифрового формату відбувається перетворення функції неперервних змінних до функцій дискретних змінних, за допомогою яких початкові змінні можна відновити з заданою точністю, та розбиття діапазону її значень на скінченну кількість інтервалів. Так неперервне зображення розділяється на прямокутні області однакового розміру і отримується набір значень кольорів у вигляді двовимірної матриці $d(x, y)$ [9].

Зменшити розмір зображення, якщо відоме його неперервне представлення $c(x, y)$ дуже просто – для цього потрібно здійснити перетворення, описані вище (дискретизацію), але з більшим кроком по вертикалі та горизонталі. Але знімок екрана – це цифрове зображення. В цьому випадку можна відновити неперервне зображення $c'(x, y)$. Для зміни розміру зображення достатньо обчислити значення $c'(x, y)$ у потрібних точках для отримання цифрового зображення $c(x, y)$ нового розміру.

Значення кольору у довільній точці $c'(x, y)$ можна представити як лінійну комбінацію значень кольорів $d(x, y)$ у деякому околі цієї точки:

$$c'(x, y) = \sum_{i,j} d(i, j) W(i - x) W(i - y) , \quad (1)$$

де $W(x)$ обирається в залежності від алгоритму інтерполяції, що використовується.

У випадку білінійної інтерполяції використовуються відомі чотири найближчі точки – по одній в кожному напрямі кожної осі:

$$W(x) = \begin{cases} 1 - |x| & |x| < 1 \\ 0 & otherwise \end{cases} . \quad (2)$$

У випадку бікубічної інтерполяції використовуються шістнадцять найближчих точок:

$$W(x) = \begin{cases} \frac{1}{2}(|x|^2 - 1)(|x| - 2) & |x| < 1 \\ -\frac{1}{6}(|x| - 1)(|x| - 2)(|x| - 3) & 1 < |x| < 2 \\ 0 & otherwise \end{cases} . \quad (3)$$

Для переведення кольору у градації сірого потрібно представити зображення у вигляді двовимірного масиву інтенсивностей кольорів. Кожна точка зображення зберігає інформацію про інтенсивність кольорів, що формують колір точки – синього (B), зеленого (G), червоного (R). Знайти значення точки у градації сірого можна наступним чином:

$$Y = 0.299R + 0.587G + 0.114B, \quad (4)$$

при використанні просторів кольорів YUV та YIQ,

$$Y = 0.2126R + 0.7152G + 0.0722B, \quad (5)$$

при використанні моделі HDTV [10].

Такі перетворення використовуються у алгоритмах знаходження перцептивного хешу зображень. Перцептивні хеш-алгоритми описують методи генерації хешів, які можна порівнювати між собою та зробити висновок про схожість зображень. Результат, на відміну від побайтового порівняння, в залежності від обраного перцептивного алгоритму не є чутливим до незначних змін у зображеннях, таких як невеликі зміни розміру, поворотів та незначних змін відтінків.

3.3 Обґрунтування методу розв’язання

Для виконання порівняння зображень було обрано алгоритм знаходження перцептивного хешу SimpleHash у поєднанні з обчисленням відстані Хеммінга для отримання відсотка схожості зображень.

Такий алгоритм потребує менше часу для виконання обчислень у порівнянні з побайтовим порівнянням зображень та не є чутливим до зміни зображень, що можуть виникати через зміну браузерів або дисплеїв машин, на яких відбувається виконання тестів.

Різні алгоритми пошуку перцептивних хешів зображень дають різні результати для змінених зображень (зображень зі зміненим розміром, поворотами, зміненими відтінками кольорів). Відомі наступні алгоритми:

Average Hash, dHash, Simple Hash, Discrete Cosine Transform Based Hash, Radial Variance Based Hash.

Average hash, або простий перцептивний хеш, оснований на середніх значеннях кольорів та є найбільш простим у реалізації. Цей алгоритм є чутливим до зміни відтінків зображення, а також дає великий відсоток невірних результатів.

dHash базується на відстеженні градієнта зображення. Цей алгоритм також є простим у реалізації. Крім того, він швидкий та дає достатньо точні результати, коли використовується у порівнянні зображень. Але при порівнянні зображень, що містять текст, dHash може дати псевдопозитивний результат, якщо текст на одному з зображень буде змінено. Тому цей алгоритм не підходить для тестування вебсторінок, адже текст є дуже важливою складовою інтерфейсів користувача.

Simple Hash базується на знаходженні низьких частот зображення. Саме низькі частоти показують структуру зображення. Такий алгоритм чутливий до поворотів зображення, але його результат може залишатися незмінним при незначних змінах розміру зображення, контрастності, яскравості його кольорів. Simple Hash є достатньо швидким та простим у реалізації. Тому він чудово підходить для вирішення поставленої задачі.

У алгоритмі Discrete Cosine Transform Based Hash для обчислення хешу використовується дискретне косинусне перетворення [11]. ДКП виражає функцію або сигнал (послідовність з кінцевого числа точок даних) у вигляді суми синусоїд з різними частотами і амплітудами. Низькочастотні коефіцієнти ДКП найбільш нечутливі до маніпуляцій з зображеннями. Це відбувається тому, що більша частина інформації сигналу, як правило, зосереджена в декількох низькочастотних коефіцієнтах. Тому отриманий хеш не змінюється при виконанні поворотів, масштабуванні зображення та розмиття. При тестуванні інтерфейсу користувача є важливим розташування

елементів та їх позиціонування. Тому через те, що даний стійкий до поворотів зображень, він не підходить для вирішення поставленої задачі.

Radial Variance Based Hash базується на побудові променевого вектора дисперсії на основі перетворення Радона [12]. Потім до нього застосовується ДКП і обчислюється хеш. Цей алгоритм стійкий до маніпуляцій з зображеннями, таких як повороти, розмиття, зміна розміру.

Для порівняння отриманого хешу та визначення міри подібності зображень використовується відстань Хеммінга або нормована відстань Хеммінга. Вона визначається шляхом обчислення кількості різних за значенням позицій у двох послідовностях. Чим ближче значення відстані Хеммінга до 0, тим більш схожі зображення. Через простоту реалізації для визначення міри подібності зображень використовується цей алгоритм.

У деяких алгоритмах, як, наприклад, Radial Variance Based Hash, для визначення схожості використовується пік взаємнокореляційної функції. Це – максимальне зі всіх значень нормованої взаємнокореляційної функції на проміжку від 0 до n, де n – значення довжини обох послідовностей. Це максимальне значення знаходиться на відрізку $[0, 1]$ і чим менше його значення, тим менш схожі порівнювані зображення.

3.4 Опис методів розв’язання

Simple Hash базується на знаходженні низьких частот зображення. Саме низькі частоти показують структуру зображення. Такий алгоритм чутливий до поворотів зображення, але його результат може залишатися незмінним при незначних змінах розміру зображення, контрастності, яскравості його кольорів.

Алгоритм представлено на рисунку 3.1.

Рисунок 3.1 – Алгоритм порівняння зображень

Дії алгоритму наступні: спочатку потрібно зменшити розмір зображення. Це потрібно для того, аби позбутися високих частот.

Зображення може зменшуватися до розміру 32 на 32 або 8 на 8. Для забезпечення найбільш достовірного результату при порівнянні зображень, використовуються зображення з розміром 32 на 32.

Наступний крок – переведення кольорів зображення до градацій сірого. Таким чином хеш зменшується у три рази, що забезпечує можливість швидко порівнювати отримані послідовності. Обчислення значення кольору пікселя у градаціях сірого Y виконується за наступною формулою:

$$Y = \frac{R+G+B}{3}, \quad (6)$$

де R – значення каналу R (червоний) пікселя;

G – значення каналу G (зелений) пікселя;

B – значення каналу B (синій) пікселя.

Після обчислення значень пікселів у градаціях сірого, потрібно обчислити середнє значення A кольору для всіх пікселів. У нашому випадку їх 1024:

$$A = \frac{1}{1024}\sum_{i=1}^{32}\sum_{j=1}^{32} Y_{i,j}. \quad (7)$$

Отримані значення використовуються для побудови хешу. Для кожного пікселя отримане значення кольору у градаціях сірого порівнюється з середнім значенням кольору для всього зображення. Якщо значення $Y_{i,j}$ для пікселя i, j більше за A, до хеш послідовності дописують число 1, в іншому випадку – 0.

Обчислення хешу виконується для обох зображень. Після виконання обчислень, отримуємо дві бінарні послідовності довжиною 1024.

Для порівняння двох бінарних послідовностей використовується нормована відстань Хеммінга. Відстань Хеммінга можна визначити як:

$$\Delta(x, y) = \sum_{x_i \neq y_i} 1, \quad i = 1, \dots, n, \quad (8)$$

де x, y – скінченні бінарні послідовності;

x_i, y_i – члени бінарних послідовностей з порядковим номером i;

n – довжина послідовності, що у нашому випадку дорівнює 1024.

Нормована відстань Хеммінга визначається наступним чином:

$$\Delta_n(x, y) = \frac{1}{n}\sum_{x_i \neq y_i} 1, \quad i = 1, \dots, n. \tag{9}$$

Нормована відстань Хеммінга знаходиться на відрізку $[0, 1]$. Чим ближче значення відстані Хеммінга до 0, тим більш схожі зображення.

Висновок до розділу

У цьому розділі дано змістовну та математичну постановку задачі, описані методи розв’язання, обґрунтовано вибір алгоритму для розв’язання задачі та наведено його детальний опис.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для розробки програмного забезпечення використовувались програмні засоби, які наведені нижче.

а) Операційна система Windows 10 – операційна система, випущена у 2015 році компанією Microsoft для робочих станцій та персональних комп’ютерів [13].

б) Об'єктно-орієнтована мова програмування C# версії 7.3 – мова програмування високого рівня, розроблена компанією Microsoft як основна мова розробки додатків для платформи Microsoft .NET. [14].

в) .NET Framework версії 4.6.1 – програмна платформа, розроблена компанією Microsoft. Використовується як середовище виконання та засіб розробки.

г) Середовище розробки Visual Studio 2017 версії 15.9.7 – інтегроване середовище розробки програмного забезпечення, випущене компанією Microsoft.

д) Specflow версії 3.1 – фреймворк з відкритим кодом, що надає засоби для написання та виконання тестів за Behavior Driven Testing методологією для .NET.

е) Selenium WebDriver версії 3.141 – надає засоби для керування браузерами. Для розробки даного програмного забезпечення використовуються вебдрайвери для браузерів Google Chrome, Opera, Internet Explorer, Mozilla Firefox – відповідно ChromeDriver, OperaDriver, IEDriver, GeckoDriver.

ж) EntityFramework 6.4 – об’єктно-орієнтована технологія доступу до даних, розроблена компанією Microsoft для .NET Framework.

з) Microsoft Office Word 2007 – програмне забезпечення для набору, редагування та оформлення тексту, випущене компанією Microsoft.

и) Sparx Systems Enterprise Architect 12 – програмне забезпечення для візуального моделювання та проектування, розроблене компанією Sparx Systems.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Вимоги до технічного забезпечення відповідають вимогам для встановлення та використання інтегрованого середовища розробки Visual Studio 2017 та браузерів, на яких виконується тестування. Мінімальні вимоги описані в таблиці 4.1.

Таблиця 4.1 – Мінімальні вимоги до технічного забезпечення

Підтримувані операційні системи	Windows 10 версії 1507 або вище, Windows 8.1 (з оновленням 2919355), Windows 7 SP1 (з останніми оновленнями Windows).
Обладнання	Процесор з тактовою частотою 1.8 GHz або вище. 2 GB оперативної пам’яті, рекомендовано – 4 GB оперативної пам’яті. Від 20-50 До 130 GB вільного місця на жорсткому диску. ОС Windows та Visual Studio рекомендується встановлювати на SSD. Відеоадаптер з мінімальним дозволом 720p (1280 на 720 пікселів). Рекомендовано – 1366 на 768 пікселів або вище.

Для використання фреймворку потрібно встановити середовище для розробки та виконання Visual Studio 2017 або вищої версії та розширення Specflow for Visual Studio 2017 для написання тестових сценаріїв за Behavior

Driven Testing методологією. Фреймворк написано на мові C# версії 7.3. Також встановлюються наступні NuGet пакети:

а) EntityFramework для роботи з базою даних;

б) Selenium.WebDriver для роботи з Selenium WebDriver API, дозволяє керувати браузерами, імітуючи дії звичайного користувача;

в) Selenium.WebDriver.ChromeDriver для роботи з браузером Google Chrome. Остання версія драйвера для цього браузера є сумісною з останньою версією браузера;

г) Selenium.WebDriver.GeckoDriver для роботи з браузером Mozilla Firefox. Остання версія драйвера для цього браузера є сумісною з останньою версією браузера;

д) Selenium.WebDriver.IEDriver для роботи з браузером Internet Explorer. Остання версія драйвера для цього браузера є сумісною з останньою версією браузера;

е) Selenium.Opera.WebDriver для роботи з браузером Opera. Остання версія драйвера для цього браузера є сумісною з останньою версією браузера;

ж) SpecRun.Runner для запуску тестів на виконання;

з) FluentAssertions для виконання перевірок.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

Структура класів поділена на рівні: Common, Pages, Steps та Tests.

Common містить класи, необхідні для визначення конфігурації виконання тестів: браузер та розмір екрану. На цьому рівні визначено клас CurrentPreferences, який описує поточні властивості конфігурації за допомогою перелічуваних типів DriverTypes, ScreenSizes та класу Size, що наслідує клас System.Attribute, які також знаходяться на цьому рівні. Діаграма класів для цього рівня показана на рисунку 4.1.

Рисунок 4.1 – Діаграма класів для рівня Common

Pages містить опис сторінок, з якими може працювати вебдрайвер для виконання автоматизованих сценаріїв. Для створення цього рівня було використано шаблон проектування PageFactory. PageFactory – це удосконалена версія PageObject, що є досить популярним шаблоном для автоматизації тестування. Він дозволяє представити вебсторінки та їх елементи у вигляді об’єктів. Так, для кожної сторінки створюється клас, що містить властивості – елементи сторінки, та методи для взаємодії з ними. Перевага цього шаблону в тому, що при зміні елементів інтерфейсу тестувальнику потрібно замінити відповідні елементи лише у класі сторінки, а не у кожному тесті, де ці елементи використовуються. Діаграма класів для цього рівня показана на рисунку 4.2.

Рисунок 4.2 – Діаграма класів для рівня Pages

На рівні Steps описані класи, що використовуються у тестах для взаємодії зі сторінками та класи-утиліти для роботи з базою даних та зображеннями. Для роботи з базою даних використовується об'єктно-орієнтована технологія Entity Framework, що дозволяє асоціювати набори даних з певними об'єктами. Класи сутностей, що відповідають таблицям бази даних, також представлені на цьому рівні. Для взаємодії зі сторінками створюються класи, що містять методи виконання повторюваних дій, таких як вибір елемента зі списку, перехід по елементам меню. Діаграма класів для цього рівня показана на рисунку 4.3.

Рисунок 4.3 – Діаграма класів для рівня Steps

На рівні Tests описуються класи з імплементацією кроків тестових сценаріїв. Також на цьому рівні міститься клас з описом дій, що мають бути

виконані до та після виконання всіх сценаріїв, наборів сценаріїв та окремих тестових сценаріїв. Це підрахунок часу виконання, генерація звіту, запис результатів до бази даних. Також визначено клас для роботи з вебдрайвером. Вебдрайвер надає можливість керувати браузером та взаємодіяти з елементами сторінки. Він імітує дії користувача, такі як натискання, ввід тексту, зміна масштабу та інші. Діаграма класів для цього рівня показана на рисунку 4.4.

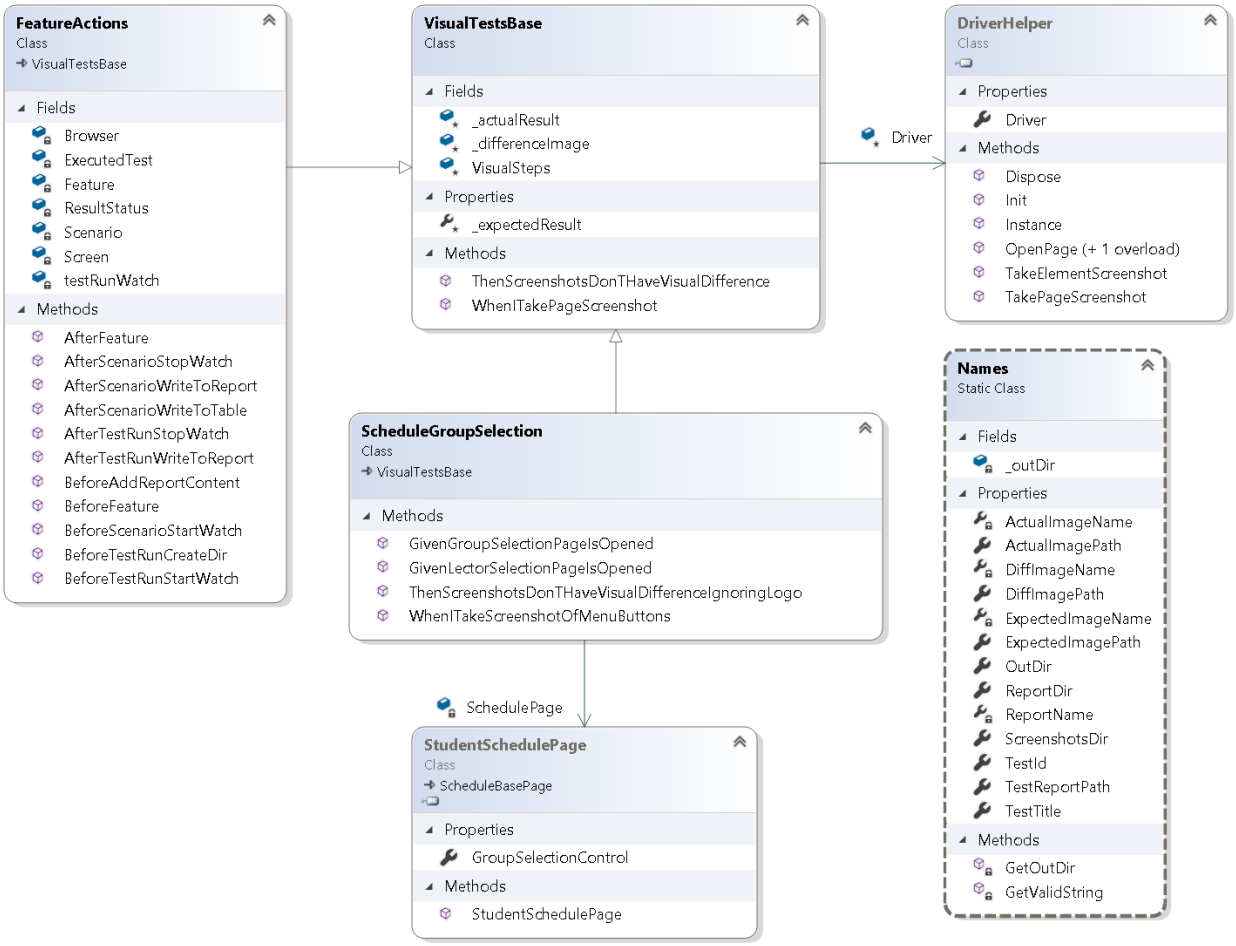


Рисунок 4.4 – Діаграма класів для рівня Tests

4.3.2 Діаграма послідовності

Коли користувач запускає виконання тестів, спочатку виконуються методи, де описані передумови виконання всіх тестів – це створення папок для результатів тестів (знімків екрану та звіту), створення структури HTML-звіту та фіксація часу початку виконання тестів. Перед виконанням кожного

набору тестів також виконуються дії. Це – створення об’єкту драйвера з описаними у класі конфігурації властивостями (розмір екрану, тип браузера) та його ініціалізація, тобто відкриття браузера. Перед виконанням конкретного тестового сценарію фіксується час початку його виконання.

Після виконання передумов відбувається виконання кожного кроку тестового сценарію, імплементація яких описана у спеціальних класах з атрибутом Binding. Для кожного кроку сценарію описується окремий метод, у яких можуть викликатися методи для взаємодії з елементами вебсторінок, створення знімків екрану, порівняння зображень з класів рівня Steps.

Як тільки було виконано останній крок тестового сценарію або у випадку, якщо при виконанні сценарію виникла помилка, запускається виконання післяумов тесту, де фіксується час завершення тесту, збереження назви тесту, назви набору тестів, до якого він належить, статусу завершення тесту, повідомлення про помилку, якщо така виникла, що потім додається до звіту та бази даних. Після виконання набору тестових сценаріїв закриваються усі вікна браузера. Після виконання усіх тестів фіксується час завершення виконання та записується загальні результати по виконанню тестів до звіту. Діаграма послідовності виконання тестів зображена на рисунку 4.5.

Рисунок 4.5 – Діаграма послідовності

4.3.3 Діаграма компонентів

Компонент Common зберігає можливі варіанти конфігурації та поточну конфігурацію, з якою виконуються тести.

Компонент Pages містить класи сторінок з описом їх елементів та методи взаємодії з елементами. Для взаємодії з елементами сторінок використовується Selenium WebDriver. Selenium WebDriver надає інтерфейс для керування браузером та його реалізацію для найбільш популярних браузерів. Selenium WebDriver дозволяє керувати браузером, імітуючи дії звичайного користувача: натискання, масштабування, наведення миші, ввід тексту, створення знімків екрана та інші. Також він надає можливість отримувати дані про елемент, такі як розташування, стан, атрибути елемента.

Компонент Steps описує більш складні методи взаємодії з сторінками, які складаються з методів, описаних у компоненті Pages, а також методи запису результатів виконання тестів до бази даних. За допомогою системи управління реляційними базами даних Microsoft SQL Server відбувається запис результатів тестування до БД. Також цей компонент містить методи порівняння зображень.

Компонент Tests містить тестові сценарії, їх імплементацію та методи передумов і післяумов виконання всіх тестів, наборів тестів або окремих тестових сценаріїв. Для імплементації кроків тестових сценаріїв використовуються методи компоненти Steps.

Рисунок 4.6 – Діаграма компонентів

4.3.4 Специфікація функцій

Опис функцій класу PageFactory наведено в таблиці 4.2.

Таблиця 4.2 – Специфікація методів класу PageFactory

Функція	Опис
public static void Init(T page, IWebDriver driver)	Ініціалізація елементів сторінки page. Вказується драйвер, що буде використовуватися для взаємодії з елементами сторінки.

Опис функцій класу ScheduleBasePage наведено в таблиці 4.3.

Таблиця 4.3 - Специфікація методів класу ScheduleBasePage

Функція	Опис
public void OpenSessionSchedule()	Натискання на кнопку «Розклад сесії».
public void OpenStudentSchedule()	Натискання на кнопку «Розклад занять».
public void OpenLecturerSchedule()	Натискання на кнопку «Розклад для викладачів».

Опис функцій класу HammingDistance наведено в таблиці 4.4.

Таблиця 4.4 – Специфікація методів класу HammingDistance

Функція	Опис
public static double FindDistance(string hash1, string hash2)	Підрахунок відстані Хеммінга між двома послідовностями у відсотках.

Опис функцій класу ScreenshotComparer наведено в таблиці 4.5.

Таблиця 4.5- Специфікація методів класу ScreenshotComparer

Функція	Опис
public static double CompareScreenshotsSimpleHash(string img1, string img2, string resultImage)	Пошук розбіжностей між двома зображеннями з іменами img1, img2 та генерація результату порівняння – зображення, що є візуалізацією розбіжностей resultImage. Метод повертає відсоток схожості зображень.
public static double GetDifferencePercentSimpleHash(string img1, string img2)	Пошук відсотка розбіжності зображень за допомогою алгортма SimpleHash та знаходження відстані Хеммінга.

Опис функцій класу SimpleHash наведено в таблиці 4.6.

Таблиця 4.6 - Специфікація методів класу SimpleHash

Функція	Опис
public static string Calculate(Image image)	Пошук перцептивного хеша для зображення image.

Опис функцій класу DriverHelper наведено в таблиці 4.7.

Таблиця 4.7 - Специфікація методів класу DriverHelper

Функція	Опис
public static DriverHelper Instance()	Повертає об’єкт драйвера, якщо він був створений, в іншому випадку створює новий.
public void Init(DriverTypes driver, ScreenSizes windowSize)	Запуск браузера driver з розмірами windowSize.
public void Dispose()	Закриття браузера.
public void OpenPage(string url)	Перехід за посиланням url.

Продовження таблиці 4.7

public void TakePageScreenshot(string fileName)	Збереження знімка сторінки під назвою fileName.
public void TakeElementScreenshot(IWebElement element, string fileName)	Збереження знімка елемента element під назвою fileName.
private IWebDriver GetDriver(DriverTypes driver)	Повертає об’єкт браузера, що відповідає типу driver.

Опис функцій класу DBHelper наведено в таблиці 4.8.

Таблиця 4.8 - Специфікація методів класу DBHelper

Функція	Опис
public static void AddTestResult(Feature feature, Scenario scenario, Browser browser, Screen screen, ResultStatus resultStatus, ExecutedTest executedTest)	Запис результату тесту до бази даних.

Опис функцій класу EnumsHelper наведено в таблиці 4.9.

Таблиця 4.9 - Специфікація методів класу EnumsHelper

Функція	Опис
public static TAttribute GetAttribute<TAttribute>(Enum value) where TAttribute : Attribute	Отримання значення атрибуту TAttribute елемента перелічуваного типу value.
public static string GetDescription(Enum value)	Отримання значення атрибуту Description елемента перелічуваного типу value.

Продовження таблиці 4.9

public static string GetHeight (Enum value)	Отримання значення атрибуту Height елемента перелічуваного типу value.
public static string GetWidth (Enum value)	Отримання значення атрибуту Width елемента перелічуваного типу value.

Опис функцій класу ImageHelper наведено в таблиці 4.10.

Таблиця 4.10 - Специфікація методів класу ImageHelper

Функція	Опис
public static Bitmap ResizeImage(Image image, int width, int height)	Змінює розмір зображення image до вказаних ширини width та висоти height.
public static int CalculateWeightByAvgOfChannels(Color c)	Підрахунок середнього значення по всім каналам кольору c.
public static string FillRectangle(string file, Rectangle rectangle)	Малює прямокутник rectangle на зображенні з розташуванням file. Це використовується для ігнорування елементів сторінки при порівнянні зображень.
public static void DrawDifferenceImage(string file1, string file2, string resultfile)	Генерація зображення з візуалізацією розбіжностей між зображеннями з розташуванням file1 та file2. Вихідне зображення зберігається з назвою resultfile.

Опис функцій класу TestReportHelper наведено в таблиці 4.11.

Таблиця 4.11 - Специфікація методів класу TestReportHelper

Функція	Опис
public static void WriteToReport(string fileName)	Запис згенерованого вмісту звіту до файлу з розташуванням fileName.

Продовження таблиці 4.11

public static void CreateStructure()	Генерація структури звіту – стилів та тегів основних елементів.
public static void AddSummary()	Запис загальних результатів виконання тестів до звіту.
public static void AddTestSummary(string testName, string result, string error, TimeSpan duration, string actualResult, string expectedResult, string diffImage)	Запис інформації про виконаний тест з назвою testName, результатом result, повідомленням про помилку error, тривалістю duration, знімками екрану actualResult та expectedResult, візуалізацією розбіжностей знімків екрану diffImage до звіту.

Опис функцій класу FeatureActions наведено в таблиці 4.12.

Таблиця 4.12 - Специфікація методів класу FeatureActions

Функція	Опис
public static void BeforeTestRunCreateDir()	Створення папок для результатів тестування перед виконанням всіх тестів.
public static void BeforeAddReportContent()	Створення файлу звіту та його структури.
public static void BeforeTestRunStartWatch()	Фіксація часу початку виконання всіх тестів.
public static void AfterTestRunStopWatch()	Фіксація часу завершення виконання всіх тестів.
public static void AfterTestRunWriteToReport()	Запис до звіту загальної інформації про виконання всіх тестів.
public static void BeforeFeature()	Запуск браузера перед початком виконання тестового набору.
public static void AfterFeature()	Закриття браузера після виконання тестового набору.

Продовження таблиці 4.12

public static void BeforeScenarioStartWatch()	Фіксація часу початку виконання тестового сценарію.
public static void AfterScenarioStopWatch()	Фіксація часу завершення виконання тестового сценарію.
public static void AfterScenarioWriteToReport()	Запис інформації про виконаний тест до звіту.
public static void AfterScenarioWriteToTable()	Запис інформації про виконаний тест до бази даних.

4.4 Опис звітів

По завершенню виконання тестових сценаріїв користувач може переглянути звіт з виконаних тестів. Звіт являє собою HTML-сторінку, яка генерується у ході виконання тестів.

Звіт містить дані про конфігурацію та результати виконання тестів і складається з наступних елементів:

а) заголовок звіту;

б) інформація про конфігурацію;

в) загальні результати виконання набору тестів;

г) деталізований опис результату виконання кожного тесту.

Загальний вигляд звіту з виконання тестів представлено на рисунку 4.7.

Test Report from 28.05.2020 22:45:50

Chrome, 1024x738

Summary

Tests run	4
Passed	3
Failed	1
Skipped	0
Success rate	75%
Total time	00:00:34.5332950

Tests

1. Full group selection page

Result	Error	Duration
OK	-	00:00:02.1207557

▶ Screenshots

2. Menu buttons

Result	Error	Duration
OK	-	00:00:01.0445180

▶ Screenshots

Рисунок 4.7 – Звіт з виконання тестів

Заголовок звіту містить дату та час генерації звіту.

В описі конфігурації вказується назва браузера та розмір екрану, з якими виконувалися тести.

Опис загальних результатів виконання набору тестів включає таблицю з наступними відомостями: кількість виконаних тестів, кількість успішно виконаних тестів, кількість пропущених тестів. Також показується відсоток успішності виконання тестів – відношення кількості успішно виконаних тестів до загальної кількості тестів та час, витрачений на виконання тестів, у форматі гг:хх:сс.чч, де компонент гг - години, вимірювані в 24-годинному форматі, хх - хвилини, сс - секунди, а чч - частки секунди.

Детальний опис кожного виконаного тесту приводиться у порядку їх виконання. Опис тесту включає назву набору тестових сценаріїв, до якого входить тест, назву тесту, результат виконання – статус завершення тесту та

повідомлення про помилку, час виконання тесту у форматі гг:хх:сс.чч, де компонент гг - години, вимірювані в 24-годинному форматі, хх - хвилини, сс - секунди, а чч - частки секунди. Також приводяться скріншот очікуваного результату тесту, скріншот, отриманий при виконанні тесту та зображення, що є результатом порівняння цих скріншотів. Вигляд тестів у звіті з різними результатами виконання представлено на рисунках 4.8 та 4.9.

Рисунок 4.8 – Успішно пройдений тест у звіті

Рисунок 4.9 – Неуспішно пройдений тест у звіті

Висновок до розділу

У цьому розділі описано засоби розробки програмного забезпечення, наведено загальні вимоги до технічного забезпечення. Наведені діаграми класів, послідовності, компонентів та їх опис. Описано звітні форми. Наведено специфікацію функцій класів.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Задачами розробки є:

а) ведення сценаріїв тестування;

б) ведення очікуваних результатів для сценаріїв тестування;

в) виконання тестових сценаріїв;

г) ведення результатів виконання сценаріїв тестування;

д) формування звітних форм.

Для створення сценарію тестування користувач спочатку створює файл з розширенням feature та додає його до проекту Tests. Після відкриття файлу, користувач набирає назву тестового набору та текст сценарію у вигляді заголовка сценарію та його кроків латинськими літерами. Назва тестового набору починається зі слова Feature. Заголовок сценарію починається зі слова Scenario. Кожен крок сценарію повинен починатися з нового рядка. Кожен крок повинен починатися зі слова Given, When або Then. Кроки, що починаються зі слова Given, описують початкові умови сценарію, When – дії, що запускають сценарій, а Then – описують очікуваний результат, до якого приводить крок When. Для опису передумов у вигляді кроків використовується слово Background. Приклад написаного тестового сценарію наведено на рисунку 5.1.

ScheduleGroupSelection.feature

```
1  Feature: ScheduleGroupSelection
2
3  Scenario: Full group selection page
4      Given group selection page is opened
5      When I take page screenshot
6      Then Screenshots don't have visual difference
7
```

Рисунок 5.1 – Приклад тестового сценарію

Для створення імплементації користувач повинен натиснути правою кнопкою миші по тексту сценарію та обрати пункт «Generate Step Definitions» (рисунок 5.2). Користувач зберігає файл у папці проекту.

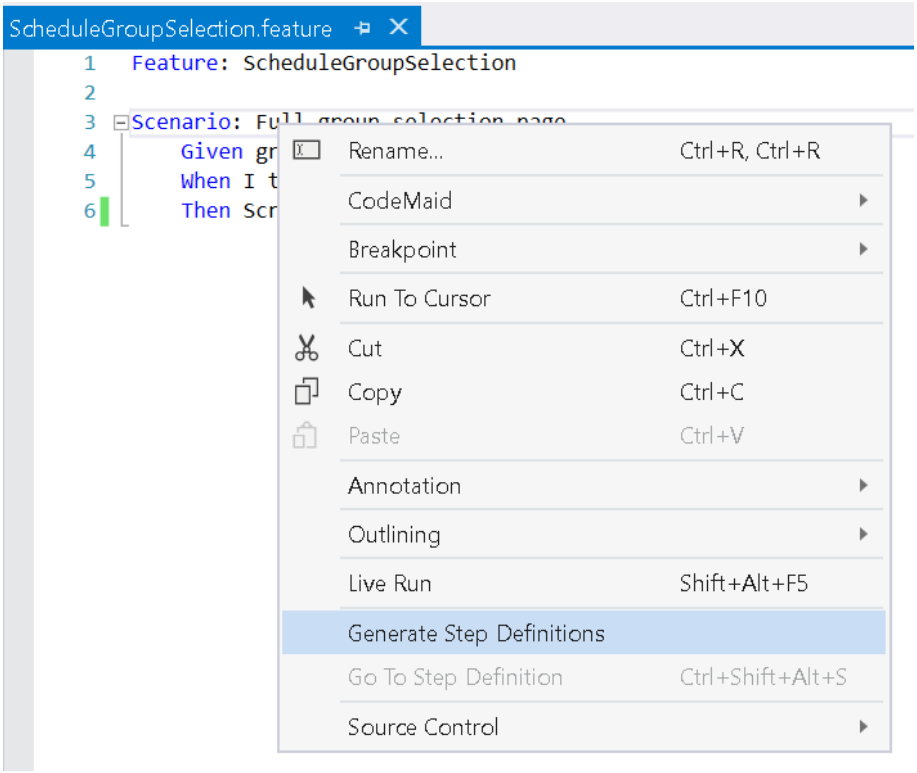


Рисунок 5.2 – Створення імплементації кроків сценарію

У створеному файлі користувач записує код реалізації кожного сценарію. Приклад показано на рисунку 5.3.

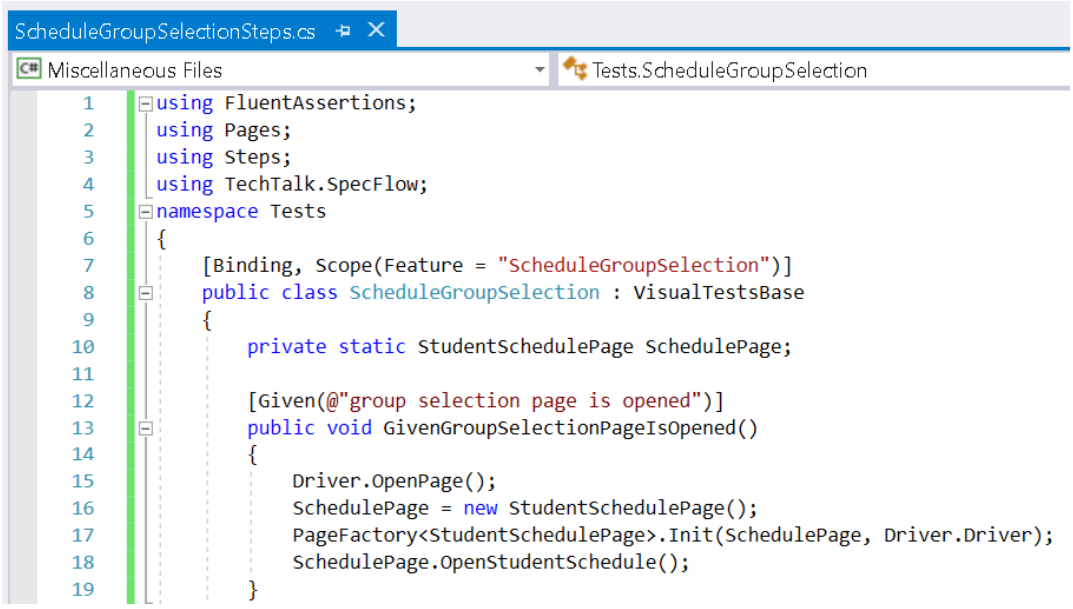


Рисунок 5.3 – Імплементація кроків сценарію

Для додавання очікуваних зображень для сценаріїв тестування користувач додає файл зображення очікуваного результату з ім'ям виду «<назва_сценарію>_expected_<розмір екрана>.png» до папки ScreenshotTesting\Tests\TestData\Screenshots.

Після цього користувач має додати файли зображень до проекту. Для цього у вікні Solution Explorer у Visual Studio потрібно знайти папку ScreenshotTesting\Tests\TestData\Screenshots, натиснути на неї правою кнопкою миші та обрати «Add», «Add Existing Item» і у переглядачі файлів знайти додані зображення. Після додавання зображень, для кожного зображення у вікні властивостей потрібно означити, що файл повинен копіюватися до папки результатів тестів при кожному виконанні тестів. Для цього для властивості «Copy To Output Directory» потрібно встановити «Copy Always», як показано на рисунку 5.4.

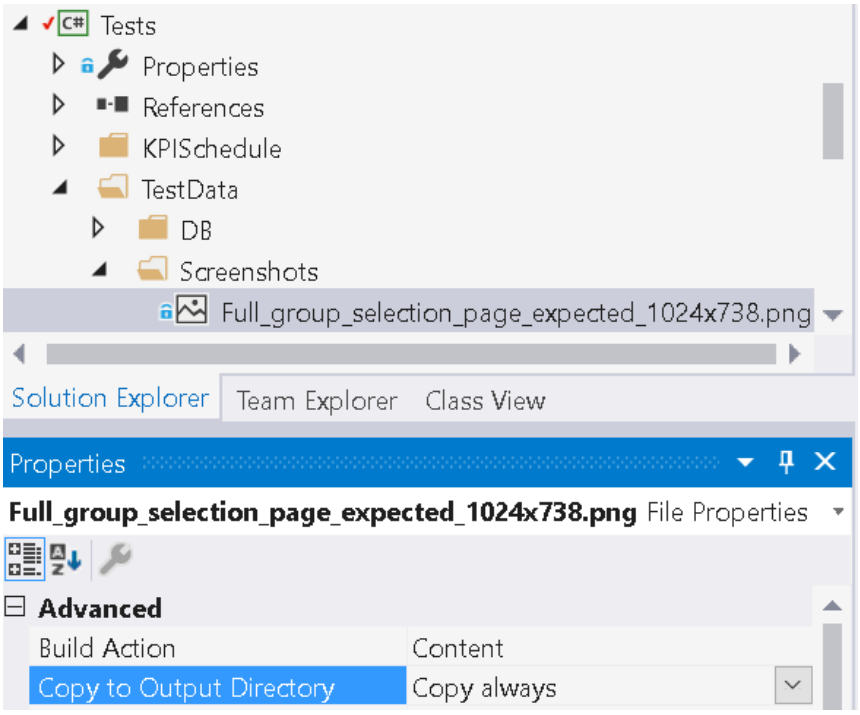


Рисунок 5.4 – Встановлення властивостей файлу

Коли користувач зберігає тестовий сценарій, його назва відображається у вікні Test Explorer у Visual Studio.

Для вибору браузера для виконання тестів користувач повинен змінити властивість класу CurrentPreferences (рисунок 5.5) driver на одне з значень перелічуваного типу DriverTypes: IE, Chrome, FireFox, Opera відповідно для браузерів Internet Explorer, Google Chrome, Mozilla Firefox, Opera.

Для вибору розміру екрана для виконання тестів користувач повинен змінити властивість класу CurrentPreferences screen на одне з значень перелічуваного типу ScreenSizes: Small, Medium, Large, що відповідають розширенням 1024 на 738, 1280 на 800, 1366 на 768. Користувач також може змінити ці значення, відредагувавши атрибути значень перелічуваного типу ScreenSizes Size та вказавши бажану ширину та довжину екрана у файлі ScreenSizes.cs (рисунок 5.6).

CurrentPreferences.cs

Common — Common.Configuration.CurrentPreferences

```
1   using Common.Enums;
2
3   namespace Common.Configuration
4   {
5       public static class CurrentPreferences
6       {
7           public static DriverTypes driver = DriverTypes.Chrome;
8           public static ScreenSizes screen = ScreenSizes.Medium;
9       }
10  }
```

Рисунок 5.5 – Налаштування умов виконання тестів

ScreenSizes.cs

Common | Common.Enums.Size

```
1   using System;
2   using System.ComponentModel;
3
4   namespace Common.Enums
5   {
6       public enum ScreenSizes
7       {
8           [Description("1366x768")]
9           [Size(1366, 768)]
10          Large,
11
12          [Description("1280x800")]
13          [Size(1280, 738)]
14          Medium,
15
16          [Description("1024x738")]
17          [Size(1024, 738)]
18          Small
19      }
```

Рисунок 5.6 – Перелічуваний тип ScreenSizes

Для запуску всіх тестів потрібно натиснути кнопку Run All. Приклад вигляду виконаних тестів у вікні Test Explorer наведено на рисунку 5.7.

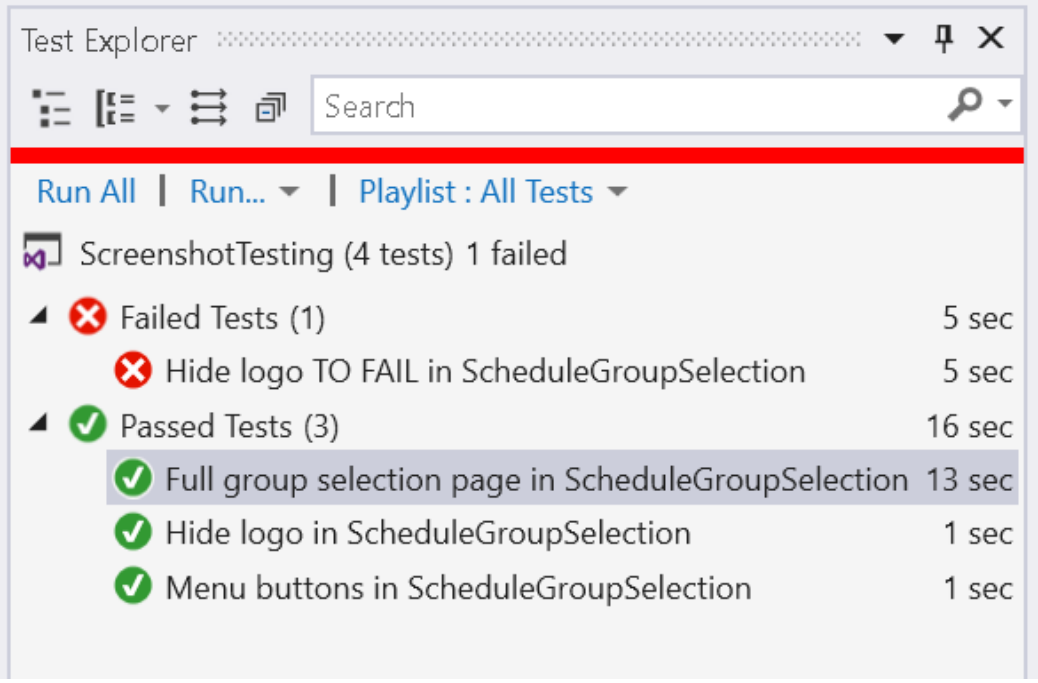


Рисунок 5.7 – Виконані тести

Після завершення тестів у папці ScreenshotTesting\TestResults створюється папка з назвою у вигляді «<ім'я_кристувача>_<дата_і_час_створення>». У папці створюються папки Report та Screenshots.

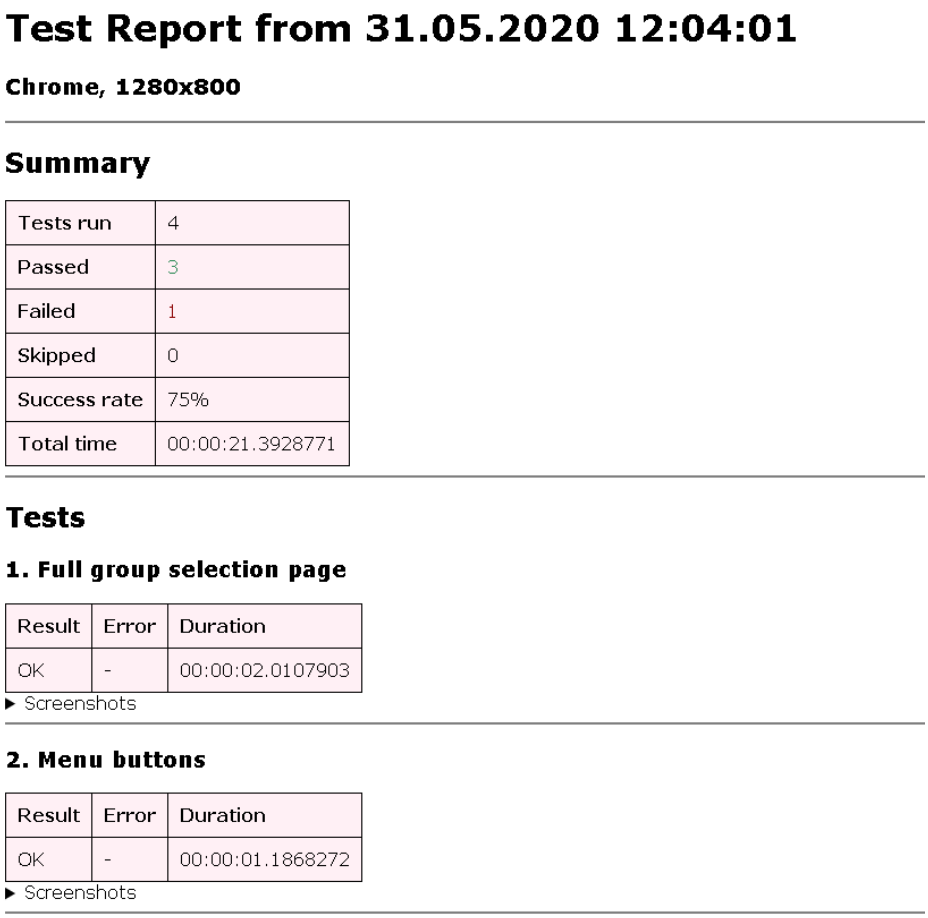
У папці Screenshots зберігаються отримані під час виконання тестів знімки екрана сторінок або їх елементів та візуалізації розбіжностей зображень очікуваного та отриманого знімків екрана, якщо ці зображення відрізняються. Знімки екрана, отримані під час виконання тестів, мають ім'я у вигляді «<назва тестового сценарію>_actual_<дата та час виконання>.png», а зображення розбіжностей – «<назва тестового сценарію>_diff_<дата та час виконання>.png» (рисунок 5.8).

Рисунок 5.8 – Отримані зображення після виконання тестів

У папці Report зберігається файл звіту – HTML-сторінка із назвою у вигляді «TestReport_<дата та час виконання>.html» (рисунок 5.9).

Рисунок 5.9 – Звіт з виконання тестів

Звіт містить дані про конфігурацію та результати виконання тестів і складається з наступних елементів: заголовок звіту, інформація про конфігурацію, загальні результати виконання набору тестів, деталізований опис результату виконання кожного тесту. Приклад звіту показано на рисунку 5.10.



Test Report from 31.05.2020 12:04:01

Chrome, 1280x800

Summary

Tests run	4
Passed	3
Failed	1
Skipped	0
Success rate	75%
Total time	00:00:21.3928771

Tests

1. Full group selection page

Result	Error	Duration
OK	-	00:00:02.0107903

▶ Screenshots

2. Menu buttons

Result	Error	Duration
OK	-	00:00:01.1868272

▶ Screenshots

Рисунок 5.10 – Звіт з виконання тестів

5.2 Випробування програмного продукту

У ході випробувань перевіряються наступні функції:

а) написання тестових сценаріїв;

б) редагування тестових сценаріїв;

в) розробка імплементації тестових сценаріїв;

г) ведення очікуваних результатів;

д) отримання знімків екрана сторінки;

е) отримання знімків екрана елементів сторінки;

ж) порівняння зображень;

з) порівняння зображень з ігноруванням елементів;

и) виконання тестових сценаріїв на різних браузерах;

к) виконання тестових сценаріїв на різних розмірах екрана;

л) отримання звіту з виконаних тестів.

Для перевірки роботи наведених вище функцій розроблено ряд тестових сценаріїв. Опис тестових сценаріїв наведено в таблиці 5.1. Пріоритет тестових сценаріїв відповідає пріоритетам функціональних вимог.

Таблиця 5.1 – Тестові сценарії

Тестові випадки у порядку виконання	Функціональна вимога	Кроки відтворення	Очікуваний результат
Написати тестовий сценарій	FR 1, FR 1.1, FR 1.2	Створити файл <назва тестового набору>.feature та відкрити його. Ввести текст тестового сценарію, починаючи кожен крок з нового рядка. Зберегти файл.	Файл збережено. Текст збереженого файлу відповідає введеному.
Редагувати тестовий сценарій.	FR 2.	Створити файл <назва тестового набору>.feature та відкрити його. Ввести текст тестового сценарію, починаючи кожен крок з нового рядка. Зберегти файл. Відкрити файл та редагувати його вміст. Зберегти файл.	Файл збережено. Текст збереженого файлу відповідає тексту після редагування.

Продовження таблиці 5.1

Створити імплементацію тестових сценаріїв.	FR 3.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм. Кроки: Правою кнопкою миші натиснути на текст тестового сценарію та обрати «Generate steps definition». У діалоговому вікні обрати «Generate». Зберегти файл. Відкрити згенерований файл. Ввести інструкції дій для кожного кроку сценаріїв на мові C#.	Файл <назва тестового набору>Steps.cs збережено. При натисканні правою кнопкою миші на крок тестового сценарію та виборі «Go To Step Definition» відкривається файл з кодом для цього кроку.
Додати зображення очікуваних результатів.	FR 4, FR 4.1.	Зберегти зображення у папці ScreenshotTesting\TestData\Screenshots. Додати збережені файли до проекту Tests.	Після початку виконання тестів файл відображається у папці ScreenshotTesting\TestResults\TestData\Screenshots

Продовження таблиці 5.1

Видалити зображення очікуваних результатів.	FR 4, FR 4.2.	Видалити існуюче зображення з папки ScreenshotTesting\TestData\Screenshots. Видалити зображення з файлів проекту Tests.	Після початку виконання тестів файл не відображається у папці ScreenshotTesting\TestResults\TestData\Screenshots
Створити знімок екрана сторінки.	FR 5, FR 5.1, FR 5.2	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: До файлу з тестовим сценарієм додати крок «When I take page screenshot». Запустити виконання тесту та дочекатися його завершення.	Файл зображення сторінки збережено з назвою «<назва тестового сценарію>_actual_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення>\Screenshots.

Продовження таблиці 5.1

Створити знімок екрана елемента.	FR 5, FR 5.3	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: До файлу з тестовим сценарієм додати крок «When I take element screenshot». У файлі з імплементацією кроків створити метод для кроку «When I take element screenshot» та викликати в ньому метод Driver.TakeElementScreenshot(<об’єкт елемента>, _actualResult) Запустити виконання тесту та дочекатися його завершення.	Файл зображення елемента збережено з назвою «<назва тестового сценарію>_actual_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення>\Screenshots.

Продовження таблиці 5.1

Порівняти однакові зображення.	FR 6, FR 6.1, FR 6.2.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Зберегти два однакові зображення. Викликати метод VisualSteps.FindDiffPercent(<шлях до зображення 1>, <шлях до зображення 2>, _differenceImage) у коді з імплементацією тестових кроків. У режимі налагодження отримати значення, що повернув метод.	Отримане значення дорівнює 0.

Продовження таблиці 5.1

Порівняти зображення з розбіжністю у розмірі до 15%.	FR 6, FR 6.1, FR 6.2.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Зберегти два однакові зображення, одне з яких більше або менше другого на 10%. Викликати метод VisualSteps.FindDiffPercent(<шлях до зображення 1>, <шлях до зображення 2>, _differenceImage) у коді з імплементацією тестових кроків. У режимі налагодження отримати значення, що повернув метод.	Отримане значення дорівнює 0.

Продовження таблиці 5.1

Порівняти різні зображення.	FR 6, FR 6.1, FR 6.2.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Зберегти два різні зображення. Викликати метод VisualSteps.FindDiffPercent(<шлях до зображення 1>, <шлях до зображення 2>, _differenceImage) у коді з імплементацією тестових кроків. У режимі налагодження отримати значення, що повернув метод.	Отримане значення більше за 0. Файл з візуалізацією розбіжностей збережено з назвою «<назва тестового сценарію>_diff_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення>\Screenshots, його вміст відповідає дійсності.

Продовження таблиці 5.1

Порівняти однакові зображення, ігноруючи елемент.	FR 6, FR 6.3.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Зберегти два однакові зображення. Викликати метод VisualSteps.FindDiffPercentIgnoreElement(<шлях до зображення 1>, <шлях до зображення 2>, _differenceImage, <об’єкт елемента>) у коді з імплементацією тестових кроків. У режимі налагодження отримати значення, що повернув метод.	Отримане значення дорівнює 0.

Продовження таблиці 5.1

Порівняти різні зображення, ігноруючи елемент.	FR 6, FR 6.4.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Зберегти два різні зображення. Викликати метод VisualSteps.FindDiffPercentIgnoreElement(<шлях до зображення 1>, <шлях до зображення 2>, _differenceImage, <об’єкт елемента>) у коді з імплементацією тестових кроків. У режимі налагодження отримати значення, що повернув метод.	Отримане значення більше за 0. Файл з візуалізацією розбіжностей збережено з назвою «<назва тестового сценарію>_diff_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення>\Screenshots, його вміст відповідає дійсності. Елемент на отриманому зображенні та у файлі розбіжностями зафарбовано чорним кольором.

Продовження таблиці 5.1

Запустити тести на браузері Google Chrome.	FR 7, FR 7.1, FR 7.1.1.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: У класі CurrentPreferences встановити driver = DriverTypes.Chrome. Запустити тести засобами Visual Studio.	На початку виконання тесту відкривається вікно браузера Google Chrome. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.
Запустити тести на браузері Internet Explorer.	FR 7, FR 7.1, FR 7.1.1.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: У класі CurrentPreferences встановити driver = DriverTypes.IE. Запустити тести засобами Visual Studio.	На початку виконання тесту відкривається вікно браузера Internet Explorer. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.

Продовження таблиці 5.1

Запустити тести з розміром екрана 1280 на 800 пікселів.	FR 7.2, FR 7.2.1.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: У класі CurrentPreferences встановити screen = ScreenSizes.Medium. Запустити тести засобами Visual Studio.	На початку виконання тесту відкривається вікно браузера розміром 1280 на 800. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.
Запустити тести з розміром екрана 1024 на 728 пікселів.	FR 7.2, FR 7.2.1.	Передумови: Створено файл <назва тестового набору>.feature з тестовим сценарієм та файл з його імплементацією. Кроки: У класі CurrentPreferences встановити screen = ScreenSizes.Small. Запустити тести засобами Visual Studio.	На початку виконання тесту відкривається вікно браузера розміром 1024 на 738. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.

Продовження таблиці 5.1

Переглянути звіт по виконаним тестам.	FR 8.	Передумови: Створено тести. Деякі тести завершуються успішно, деякі – ні. Кроки: Запустити тести та дочекатися завершення їх виконання.	Файл звіту збережено з назвою «TestReport_<дата та час виконання>.html» у папці ScreenshotTesting\TestResults\<ім'я користувача>_<час створення>\Report, його вміст відповідає дійсності. Звіт має наступні елементи: заголовок звіту, інформація про конфігурацію, загальні результати виконання набору тестів, деталізований опис результату виконання кожного тесту.

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач автоматизованого тестування інтерфейсу користувача вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Результати випробувань наведено в таблиці 5.2.

Таблиця 5.2 – Результати випробувань

Тестові випадки	Отримані результати	Результат випробування
Написати тестовий сценарій	Файл збережено. Текст збереженого файлу відповідає введеному.	Випробування пройдене успішно.
Редагувати тестовий сценарій.	Файл збережено. Текст збереженого файлу відповідає тексту після редагування.	Випробування пройдене успішно.
Створити імплементацію тестових сценаріїв.	Файл <назва тестового набору>Steps.cs збережено. При натисканні правою кнопкою миші на крок тестового сценарію та виборі «Go To Step Definition» відкривається файл з кодом цього кроку.	Випробування пройдене успішно.

Продовження таблиці 5.2

Додати зображення очікуваних результатів.	Після початку виконання тестів файл відображається у папці ScreenshotTesting\TestResults\TestData \Screenshots	Випробування пройдене успішно.
Видалити зображення очікуваних результатів.	Після початку виконання тестів файл не відображається у папці ScreenshotTesting\TestResults\TestData \Screenshots	Випробування пройдене успішно.
Створити знімок екрана сторінки.	Файл зображення сторінки збережено з назвою «<назва тестового сценарію>_actual_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення> \Screenshots.	Випробування пройдене успішно.
Створити знімок екрана елемента.	Файл зображення елемента збережено з назвою «<назва тестового сценарію>_actual_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення> \Screenshots.	Випробування пройдене успішно.
Порівняти однакові зображення.	Отримане значення дорівнює 0.	Випробування пройдене успішно.
Порівняти зображення з розбіжністю у розмірі до 15%.	Отримане значення дорівнює 0.	Випробування пройдене успішно.

Продовження таблиці 5.2

Порівняти різні зображення.	Отримане значення більше за 0. Файл з візуалізацією розбіжностей збережено з назвою «<назва тестового сценарію>_diff_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім'я користувача>_<час створення>\Screenshots, його вміст відповідає дійсності.	Випробування пройдене успішно.
Порівняти однакові зображення, ігноруючи елемент.	Отримане значення дорівнює 0.	Випробування пройдене успішно.
Порівняти різні зображення, ігноруючи елемент.	Отримане значення більше за 0. Файл з візуалізацією розбіжностей збережено з назвою «<назва тестового сценарію>_diff_<дата та час виконання>.png» у папці ScreenshotTesting\TestResults\<ім'я користувача>_<час створення>\Screenshots, його вміст відповідає дійсності. Елемент на отриманому зображенні та у файлі з візуалізацією розбіжностей зафарбовано чорним кольором.	Випробування пройдене успішно.

Продовження таблиці 5.2

Запустити тести на браузері Google Chrome.	На початку виконання тесту відкривається вікно браузера Google Chrome. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.	Випробування пройдене успішно.
Запустити тести на браузері Internet Explorer.	На початку виконання тесту відкривається вікно браузера Internet Explorer. Всі дії виконуються у цьому вікні. Після завершення тесту вікно браузера закривається.	Випробування пройдене успішно.
Запустити тести з розміром екрана 1280 на 800 пікселів.	На початку виконання тесту відкривається вікно браузера розміром 1280 на 800. Всі дії виконуються у цьому вікні.	Випробування пройдене успішно.
Запустити тести з розміром екрана 1024 на 728 пікселів.	На початку виконання тесту відкривається вікно браузера розміром 1024 на 738. Всі дії виконуються у цьому вікні.	Випробування пройдене успішно.
Переглянути звіт по виконаним тестам.	Файл звіту збережено з назвою «TestReport_<дата та час виконання>.html» у папці ScreenshotTesting\TestResults\<ім’я користувача>_<час створення> \Report, його вміст відповідає дійсності. Звіт має елементи: заголовок звіту, інформація про конфігурацію, загальні результати виконання набору тестів, опис результату кожного тесту.	Випробування пройдене успішно.

Висновок до розділу

У цьому розділі наведено керівництво користувача, описані сценарії тестування програмного забезпечення, наведено результати тестування.

ЗАГАЛЬНІ ВИСНОВКИ

В рамках дипломного проекту створено фреймворк, що полегшує роботу автоматизаторів тестування за рахунок зменшення часу на розробку та виконання тестів, підвищує ефективність тестів. В дипломному проекті запропоновано і розглянуто метод автоматизації функціональних тестів, що зменшує час роботи тестів, час, що витрачається на аналіз результатів тестування та збільшує покриття застосування тестами.

Тема роботи є актуальною оскільки тестування програмного забезпечення є одним з основним етапів робробки.

В першому розділі було проаналізовано предметне середовище, а саме процесс діяльності тестування. Сформульовано основні функції розробки у функціональній моделі. Проаналізовані наявні аналоги. Поставлені ціль та задачі розробки.

В розділі «Інформаційне забезпечення» визначені вхідні та вихідні дані, наведена структура бази даних для зберігання результатів тестування.

В розділі «Математичне забезпечення» сформульовано змістовну та математичну постановку задачі порівняння зображень, проаналізовано методи розв'язання та обгрунтовано вибір методу розв'язання – порівняння зображень за допомогою перцептивного алгоритму Simple Hash.

В розділі «Програмне та технічне забезпечення» обгрунтовані основні засоби розробки, описана архітектура рпогрманого забезпечення, специфікація основних методів.

У технологічному розділі наведено інструкцію користувача та результати випробування.

Розроблено фреймворк для розробки і виконання автоматизованих тестів. Визначені та виконані набори випробувань.

Таким чином, розроблене програмне забезпечення вирішує всі поставлені перед ним задачі та відповідає поставленим вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mike Cohn. Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional. 2009. 478 с. ISBN 978-0-321-57936-2.

2. Documentation for Selenium. Webdriver Documentation. [Електронний ресурс] — Режим доступу: https://www.selenium.dev/documentation/en/webdriver/

3. Getting Started with Behavior Driven Development. [Електронний ресурс] — Режим доступу: https://specflow.org/bdd/

4. WebDriver Screenshot utility. Take screenshots, crop, prettify, compare. [Електронний ресурс] — Режим доступу: https://github.com/pazone/ashot

5. Flue2ent - fluent e2e tests. [Електронний ресурс] — Режим доступу: http://definitylabs.org/#/products/flue2ent

6. Happo.io – Crossbrowser Screenshot Testing. [Електронний ресурс] — Режим доступу: https://happo.io/

7. ImageMagick - Convert, Edit, or Compose Bitmap Images. [Електронний ресурс] — Режим доступу: https://imagemagick.org/index.php

8. Allure Framework. [Електронний ресурс] — Режим доступу: https://docs.qameta.io/allure/

9. Получение и обработка изображений на ЭВМ : учебно-методическое пособие. В.В. Старовойтов, Ю.И. Голуб. Минск : БНТУ, 2018. 204 с.

10. Keith Jack. 2007. 976 с. Video Demystified: A Handbook for the Digital Engineer. Newnes, USA. ISBN 978-0-7506-8395-1.

11. Simple and DCT perceptual hash-algorithms. [Електронний ресурс] — Режим доступу: www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html

12. Standaert, F.X., Lefebvre, F., Rouvroy, G., Macq, B.M., Quisquater, J.J. and Legat, J.D.: Practical evaluation of a radial soft hash algorithm. In

Proceedings of the International Symposium on Information Technology: Coding and Computing (ITCC). IEEE, Apr. 2005

13. Windows 10 and Windows Server 2019 update history. [Електронний ресурс] — Режим доступу: https://support.microsoft.com/en-gb/help/4464619/windows-10-update-history

14. Introduction to the C# language and the .NET Framework. [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-gb/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Додаток А

Тексти програмного коду

Засоби автоматизованого тестування інтерфейсу користувача

(Найменування програми (документа))

DVD-R

(Вид носія даних)

23 арк, 455406 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020 року

```
using Common.Enums;

namespace Common.Configuration
{
   public static class CurrentPreferences
   {
      public static DriverTypes driver = DriverTypes.Chrome;
      public static ScreenSizes screen = ScreenSizes.Medium;
   }
}

namespace Common.Enums
{
   public enum DriverTypes
   {
      IE,
      Chrome,
      FireFox,
      Opera
   }
}

using System;
using System.ComponentModel;

namespace Common.Enums
{
   public enum ScreenSizes
   {
      [Description("1366x768")]
      [Size(1366, 768)]
      Large,

      [Description("1280x800")]
      [Size(1280, 738)]
      Medium,

      [Description("1024x738")]
      [Size(1024, 738)]
      Small
   }

   public class Size : Attribute
   {
      public int Height { get; private set; }
      public int Width { get; private set; }
```

```
            public Size(int width, int height)
            {
                Height = height;
                Width = width;
            }
        }
    }
    using OpenQA.Selenium.Support.PageObjects;
    using OpenQA.Selenium;


    namespace Pages
    {
        public static class PageFactory<T>
        {
            public static void Init(T page, IWebDriver driver)
            {
                PageFactory.InitElements(driver, page);
            }
        }
    }

    using OpenQA.Selenium;
    using OpenQA.Selenium.Support.PageObjects;

    namespace Pages
    {
        public class ScheduleBasePage
        {
            [FindsBy(How = How.Id, Using = "ctl00_lBtnSchedule")]
            public IWebElement StudentScheduleMenuItem { get; set; }

            [FindsBy(How = How.Id, Using = "ctl00_lBtnSession")]
            public IWebElement SessionScheduleMenuItem { get; set; }

            [FindsBy(How = How.Id, Using = "ctl00_lBtnLecturerSchedule")]
            public IWebElement LecturerScheduleMenuItem { get; set; }

            [FindsBy(How = How.Id, Using = "ctl00_Image1")]
            public IWebElement Logo { get; set; }

            public void OpenSessionSchedule() =>
SessionScheduleMenuItem.Click();
```

```
            public void OpenStudentSchedule() =>
StudentScheduleMenuItem.Click();
            public void OpenLecturerSchedule() =>
LecturerScheduleMenuItem.Click();
        }
    }
    using OpenQA.Selenium;
    using OpenQA.Selenium.Support.PageObjects;

    namespace Pages
    {
        public class StudentSchedulePage : ScheduleBasePage
        {
            [FindsBy(How = How.XPath, Using = ".//span[@id =
'ctl00_MainContent_ctl00_lblGroup']/parent::div/parent::div")]
            public IWebElement GroupSelectionControl { get; set; }

        }
    }

    using System.Linq;

    namespace Steps.Comparer
    {
        public static class HammingDistance
        {
            public static double FindDistance(string hash1, string hash2)
            {
                int distance =
                    hash1.ToCharArray()
                    .Zip(hash2.ToCharArray(), (c1, c2) => new { c1, c2 })
                    .Count(m => m.c1 != m.c2);
                return (double)distance / hash1.Length;
            }
        }
    }
    using ImageMagick;
    using Steps.Util;
    using System.Drawing;
    using System.IO;

    namespace Steps.Comparer
    {
        public static class ScreenshotComparer
        {
```

```
            public static double CompareScreenshotsSimpleHash(string img1,
string img2, string resultImage)
            {
               double diff = GetDifferencePercentSimpleHash(img1, img2);
               if (diff != 0)
               {
                  ImageHelper.DrawDifferenceImage(img1, img2, resultImage);
               }
               return diff;
            }

            public static double GetDifferencePercentSimpleHash(string img1,
string img2)
            {
               Image image1 = Image.FromFile(img1);
               Image image2 = Image.FromFile(img2);
               var hash1 = SimpleHash.Calculate(image1);
               var hash2 = SimpleHash.Calculate(image2);
               return HammingDistance.FindDistance(hash1, hash2) * 100;
            }
         }
      }
      using Steps.Util;
      using System.Collections.Generic;
      using System.Drawing;

      namespace Steps.Comparer
      {
         public static class SimpleHash
         {
            public static string Calculate(Image image)
            {
               Bitmap bitmap = null;
               int[,] matrix = new int[32, 32];
               float average = 0;
               bitmap = ImageHelper.ResizeImage(image, 32, 32);
               for (int x = 0; x < 32; x++)
               {
                  for (int y = 0; y < 32; y++)
                  {
                     matrix[x,                                    y]                    =
ImageHelper.CalculateWeightByAvgOfChanels(bitmap.GetPixel(x, y));
                     average += matrix[x, y];
                  }
               }
```

```
            average /= 32 * 32;

            string reslist = string.Empty;
            for (int x = 0; x < 32; x++)
            {
               for (int y = 0; y < 32; y++)
               {
                  if (matrix[x, y] >= average) reslist+=1;
                  else reslist+=0;
               }
            }

            bitmap.Dispose();
            return reslist;
         }
      }
   }
using Common.Enums;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.IE;
using OpenQA.Selenium.Opera;
using Steps.Util;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;

namespace Steps.DriverSteps
{
   public class DriverHelper
   {
      private static DriverHelper _instance;
      public IWebDriver Driver { private set; get; }
      private readonly string _url = "http://rozklad.kpi.ua/";

      private DriverHelper() { }

      public static DriverHelper Instance()
      {
         if (_instance == null)
            _instance = new DriverHelper();
         return _instance;
      }
```

```
        public void Init(DriverTypes driver, ScreenSizes windowSize)
        {
            Driver = GetDriver(driver);

            Driver.Manage().Window.Size =
                new System.Drawing.Size(
                    EnumsHelper.GetWidth(windowSize),
                    EnumsHelper.GetHeight(windowSize));
            var l = Driver.Manage().Window.Size.Height;
            var fl = Driver.Manage().Window.Size.Width;
        }

        public void Dispose()
        {
            Driver.Quit();
        }

        public void OpenPage()
        {
            Driver.Navigate().GoToUrl(_url);
        }

        public void OpenPage(string url)
        {
            Driver.Navigate().GoToUrl(url);
        }

        public void TakePageScreenshot(string fileName)
        {
            Screenshot screenshot =
((ITakesScreenshot)Driver).GetScreenshot();
            screenshot.SaveAsFile(fileName);
        }

        public void TakeElementScreenshot(IWebElement element, string
fileName)
        {
            Screenshot screenshot =
((ITakesScreenshot)Driver).GetScreenshot();
            var pageBitmap = ImageHelper.ResizeImageDefault(new
Bitmap(new MemoryStream(screenshot.AsByteArray())));
            var elementScreenshot = pageBitmap.Clone(new
Rectangle(element.Location, element.Size), pageBitmap.PixelFormat);
            elementScreenshot.Save(fileName, ImageFormat.Png);
```

```
            }

            private IWebDriver GetDriver(DriverTypes driver)
            {
               switch (driver)
               {
                  case DriverTypes.Chrome: return new
ChromeDriver(@"C:\Users\ya\Source\Repos\ScreenshotTesting\Steps\bin\Debug")
;
                  case DriverTypes.FireFox: return new FirefoxDriver();
                  case DriverTypes.IE: return new InternetExplorerDriver();
                  case DriverTypes.Opera: return new OperaDriver();
                  default: return null;
               }
            }
         }
      }
      using System.Collections.Generic;
      namespace Steps.Util.DB.Models
      {
         public class Browser
         {
            public int Id { get; set; }
            public string BrowserName { get; set; }
            public ICollection<ExecutedTest> ExecutedTests { get; set; }

            public Browser()
            {
               ExecutedTests = new List<ExecutedTest>();
            }
         }
      }

      using System;
      namespace Steps.Util.DB.Models
      {
         public class ExecutedTest
         {
            public int Id { get; set; }
            public string ErrorMessage { get; set; }
            public DateTime StartTime { get; set; }
            public DateTime EndTime { get; set; }

            public int? ScenarioId { get; set; }
            public Scenario Scenario { get; set; }
```

```
        public int? BrowserId { get; set; }
        public Browser Browser { get; set; }

        public int? ScreenId { get; set; }
        public Screen Screen { get; set; }

        public int? ResultStatusId { get; set; }
        public ResultStatus ResultStatus { get; set; }
    }
}
using System.Collections.Generic;

namespace Steps.Util.DB.Models
{
    public class Feature
    {
        public int Id { get; set; }
        public string FeatureName { get; set; }
        public ICollection<Scenario> Scenarios { get; set; }

        public Feature()
        {
            Scenarios = new List<Scenario>();
        }
    }
}
using System;
using System.Collections.Generic;

namespace Steps.Util.DB.Models
{
    public class ResultStatus
    {
        public int Id { get; set; }
        public string TestStatus { get; set; }
        public ICollection<ExecutedTest> ExecutedTests { get; set; }

        public ResultStatus()
        {
            ExecutedTests = new List<ExecutedTest>();
        }
    }
}
using System.Collections.Generic;
```

```
namespace Steps.Util.DB.Models
{
   public class Scenario
   {
      public int Id { get; set; }
      public string ScenarioName { get; set; }
      public int? FeatureId { get; set; }
      public Feature Feature { get; set; }
      public ICollection<ExecutedTest> ExecutedTests { get; set; }

      public Scenario()
      {
         ExecutedTests = new List<ExecutedTest>();
      }
   }
}
using System;
using System.Collections.Generic;
namespace Steps.Util.DB.Models
{
   public class Screen
   {
      public int Id { get; set; }
      public string Size { get; set; }
      public ICollection<ExecutedTest> ExecutedTests { get; set; }

      public Screen()
      {
         ExecutedTests = new List<ExecutedTest>();
      }
   }
}
using Steps.Util.DB.Models;

namespace Steps.Util.DB
{
   public static class DBHelper
   {
      public static void AddTestResult(Feature feature, Scenario scenario,
Browser browser,
         Screen screen, ResultStatus resultStatus, ExecutedTest executedTest)
      {
         using (TestResultContext db = new TestResultContext())
         {
```

```
                db.Features.Add(feature);
                db.SaveChanges();

                db.Scenarios.Add(scenario);
                db.SaveChanges();

                db.Browsers.Add(browser);
                db.SaveChanges();

                db.Screens.Add(screen);
                db.SaveChanges();

                db.ResultStatuses.Add(resultStatus);
                db.SaveChanges();

                db.ExecutedTests.Add(executedTest);
                db.SaveChanges();
            }
        }
    }
}
using Steps.Util.DB.Models;
using System.Data.Entity;

namespace Steps.Util.DB
{
    public class TestResultContext : DbContext
    {
        public TestResultContext() : base("TestsResults")
        { }

        public DbSet<Feature> Features { get; set; }
        public DbSet<Scenario> Scenarios { get; set; }
        public DbSet<Browser> Browsers { get; set; }
        public DbSet<Screen> Screens { get; set; }
        public DbSet<ResultStatus> ResultStatuses { get; set; }
        public DbSet<ExecutedTest> ExecutedTests { get; set; }
    }
}
using Common.Enums;
using System;
using System.ComponentModel;
using System.Linq;

namespace Steps.Util
```

```
        {
            public static class EnumsHelper
            {
                public static TAttribute GetAttribute<TAttribute>(Enum value)
                where TAttribute : Attribute
                {
                    var enumType = value.GetType();
                    var name = Enum.GetName(enumType, value);
                    return
enumType.GetField(name).GetCustomAttributes(false).OfType<TAttribute>().Sin
gleOrDefault();
                }

                public static string GetDescription(Enum value) =>
GetAttribute<DescriptionAttribute>(value).Description;
                public static int GetHeight(Enum value) =>
GetAttribute<Size>(value).Height;
                public static int GetWidth(Enum value) =>
GetAttribute<Size>(value).Width;
            }
        }
        using Common.Configuration;
        using System.Drawing;
        using System.Drawing.Drawing2D;
        using System.Drawing.Imaging;

        namespace Steps.Util
        {
            public static class ImageHelper
            {
                public static Bitmap ResizeImage(Image image, int width, int height)
                {
                    var destRect = new Rectangle(0, 0, width, height);
                    var destImage = new Bitmap(width, height);

                    destImage.SetResolution(image.HorizontalResolution,
                   image.VerticalResolution);

                    using (var graphics = Graphics.FromImage(destImage))
                    {
                        graphics.CompositingMode = CompositingMode.SourceCopy;
                        graphics.CompositingQuality = CompositingQuality.HighQuality;
                        graphics.InterpolationMode =
InterpolationMode.HighQualityBicubic;
                        graphics.SmoothingMode = SmoothingMode.AntiAlias;
```

```
                graphics.PixelOffsetMode = PixelOffsetMode.HighQuality;

                using (var wrapMode = new ImageAttributes())
                {
                    wrapMode.SetWrapMode(WrapMode.TileFlipXY);
                    graphics.DrawImage(image, destRect, 0, 0, image.Width,
                   image.Height, GraphicsUnit.Pixel, new ImageAttributes());
                }
            }

            return destImage;
        }

        public static Bitmap ResizeImageDefault(Image image)
        {
            int wigth = EnumsHelper.GetAttribute<Common.Enums.Size>(CurrentPreferences.screen).Width - 14;
            int height = EnumsHelper.GetAttribute<Common.Enums.Size>(CurrentPreferences.screen).Height - 132;
            return ResizeImage(image, wigth, height);
        }

        public static int CalculateWeightByAvgOfChanels(Color c)
        {
            return (c.R + c.G + c.B) / 3;
        }

        public static string FillRectangle(string file, Rectangle rectangle)
        {
            Image image = ResizeImageDefault(Image.FromFile(file));
            Graphics graphics = Graphics.FromImage(image);
            var brush = new SolidBrush(Color.Black);
            graphics.FillRectangle(brush, rectangle);
            string newFilePath = file.Replace(".png", "_ignored.png");
            image.Save(newFilePath);
            return newFilePath;
        }

        public static void DrawDifferenceImage(string file1, string file2, string resultfile)
        {
            Image image1 = Image.FromFile(file1);
            Image image2 = Image.FromFile(file2);
```

```
                Bitmap b1 = (Bitmap)image1;
                Bitmap b2;

                if (image2.Height != image1.Height || image2.Width != image1.Width)
                {
                    b2 = ResizeImage(image2, image1.Width, image1.Height);
                }
                else b2 = (Bitmap)image2;
                Bitmap result = new Bitmap(image1.Width, image1.Height);

                Color ne_color = Color.Red;
                for (int x = 0; x < result.Width; x++)
                {
                    for (int y = 0; y < result.Height; y++)
                    {
                        if (b1.GetPixel(x, y).Equals(b2.GetPixel(x, y)))
                        {
                            Color pixelColor = b1.GetPixel(x, y);
                            var greyValue = CalculateWeightByAvgOfChanels(pixelColor) / 2;
                            Color greyColor = Color.FromArgb(greyValue, greyValue, greyValue);
                            result.SetPixel(x, y, greyColor);
                        }
                        else
                        {
                            result.SetPixel(x, y, ne_color);
                        }
                    }
                }

                result.Save(resultfile, ImageFormat.Png);
            }
        }
    }
    using Common.Configuration;
    using System;
    using System.IO;
    using System.Text;

    namespace Steps.Util
    {
        public static class TestReportHelper
        {
```

```
        private static string _reportContent;
        private static int _totalTestsCount;
        private static int _totalPassedCount;
        private static int _totalSkippedCount;
        public static TimeSpan TotalTime { get; set; }

        public static void WriteToReport(string fileName)
        {
            FileStream fs = File.Create(fileName);
            using (var writer = new StreamWriter(fs, Encoding.UTF8))
            {
                writer.Write(_reportContent);
            }
        }

        public static void CreateStructure()
        {
            _reportContent = "<!DOCTYPE html> " +
                "<html> " +
                "<head>" +
                "<style>" +
                ".filter details + details { margin-top: 1em; } " +
                "body{ font-family: verdana;}" +
                "img { width: 100%; }" +
                "table,th,td{ border: 1px solid black; border-collapse: collapse;
background-color: LavenderBlush;}" +
                "table#screenshots{ width: 100%;  background-color: White;}" +
                "th,td{ padding:10px; text-align:left;}" +
                "</style>" +
                "</head>" +
                "<body>" +
                $"<h1>Test Report from {DateTime.Now}</h1>" +
                $"<h3>{CurrentPreferences.driver},
{EnumsHelper.GetDescription(CurrentPreferences.screen)}</h3>" +
                "<hr noshade=\"noshade\">" +
                "<h2>Summary</h2>" +
                "<hr noshade=\"noshade\">" +
                "<h2>Tests</h2>" +
                "</body>" +
                "</html>";
        }

        public static void AddSummary()
        {
```

```
            double passRate = ((double)_totalPassedCount / _totalTestsCount) *
100;
            string summary = "<h2>Summary</h2>" +
               "<table> " +
               "<tr> " +
               "<th>Tests run</th> " +
               $"<td>{_totalTestsCount}</td> " +
               "</tr> " +
               "<tr> " +
               "<th>Passed</th> " +
               $"<td style=\"color: SeaGreen; \">{_totalPassedCount}</td> " +
               "</tr> " +
               "<tr> " +
               "<th>Failed</th> " +
               $"<td style=\"color: DarkRed; \">{_totalTestsCount -
_totalPassedCount - _totalSkippedCount}</td> " +
               "</tr> " +
               "<tr> " +
               "<th>Skipped</th> " +
               $"<td>{_totalSkippedCount}</td> " +
               "</tr> " +
               "<tr> " +
               "<th>Success rate</th>" +
               $"<td>{passRate}%</td> " +
               "</tr> " +
               "<tr> " +
               "<th>Total time</th> " +
               $"<td>{TotalTime}</td> " +
               "</tr> " +
               "</table>";
            _reportContent = _reportContent.Replace("<h2>Summary</h2>",
summary);
         }

         public static void AddTestSummary(string testName, string result,
string error, TimeSpan duration,
            string actualResult, string expectedResult, string diffImage)
         {
            _totalTestsCount++;

            var testSummary = $"<h3>{_totalTestsCount}. {testName}</h3> " +
               "<table> " +
               "<tr> " +
               "<th>Result</th> " +
               "<th>Error</th> " +
```

```
                    "<th>Duration</th> " +
                    "</tr> " +
                    "<tr> " +
                    $"<td>{result}</td> " +
                    $"<td>{error}</td> " +
                    $"<td>{duration}</td> " +
                    "</tr> " +
                    "</table>" +
                    "<details>"+
                    "<summary>" +
                    "Screenshots" +
                    "</summary>"+
                    "<table id=\"screenshots\"> " +
                    "<tr> " +
                    "<th>Expected</th> " +
                    "<th>Actual</th> " +
                    "<th>Difference</th> " +
                    "</tr> " +
                    "<tr> " +
                    $"<td><img src={expectedResult}></td> " +
                    $"<td><img src={actualResult}></td> " +
                    $"<td><img src={diffImage}></td> " +
                    "</tr> " +
                    "</table>" +
                    "</details>" +
                    "<hr noshade=\"noshade\">" +
                    "</body>" +
                    "</html>";
                _reportContent = _reportContent.Replace("</body></html>",
testSummary);
                if (result.Equals("OK")) _totalPassedCount++;
                else if (result.Equals("Skipped")) _totalSkippedCount++;
            }
        }
    }
    using OpenQA.Selenium;
    using Steps.DriverSteps;
    using Steps.Comparer;
    using System.Drawing;
    using Steps.Util;

    namespace Steps
    {
        public class VisualSteps
        {
```

```
            private readonly DriverHelper driver = DriverHelper.Instance();

            public void TakeScreenshot(string file)
            {
                driver.TakePageScreenshot(file);
            }

            public double FindDiffPercent(string actualImg, string expectedImg, string resultImg)
            {
                return ScreenshotComparer.CompareScreenshotsSimpleHash(actualImg, expectedImg, resultImg);
            }

            public double FindDiffPercentIgnoreElement(string actualImg, string expectedImg, string resultImg, IWebElement element)
            {
                string actualImgIgnored = GetImageHideElement(actualImg, element);
                string expectedImgIgnored = GetImageHideElement(expectedImg, element);
                return FindDiffPercent(actualImgIgnored, expectedImgIgnored, resultImg);
            }

            private string GetImageHideElement(string file, IWebElement element)
            {
                return ImageHelper.FillRectangle(file, new Rectangle(element.Location, element.Size));
            }
        }
    }
    using FluentAssertions;
    using Pages;
    using Steps;
    using System;
    using TechTalk.SpecFlow;
    namespace Tests
    {
        [Binding, Scope(Feature = "ScheduleGroupSelection")]
        public class ScheduleGroupSelection : VisualTestsBase
        {
            private static StudentSchedulePage SchedulePage;
```

```
            [Given(@"group selection page is opened")]
            public void GivenGroupSelectionPageIsOpened()
            {
                Driver.OpenPage();
                SchedulePage = new StudentSchedulePage();
                PageFactory<StudentSchedulePage>.Init(SchedulePage,
Driver.Driver);
                SchedulePage.OpenStudentSchedule();
            }

            [Given(@"lector selection page is opened")]
            public void GivenLectorSelectionPageIsOpened()
            {
                Driver.OpenPage();
                SchedulePage = new StudentSchedulePage();
                PageFactory<StudentSchedulePage>.Init(SchedulePage,
Driver.Driver);
                SchedulePage.OpenLecturerSchedule();
            }

            [When(@"I take screenshot of menu buttons")]
            public void WhenITakeScreenshotOfMenuButtons()
            {
                _actualResult = Names.ActualImagePath;

Driver.TakeElementScreenshot(SchedulePage.GroupSelectionControl,
_actualResult);
            }

            [Then(@"Screenshots don't have visual difference ignoring logo")]
            public                                                    void
ThenScreenshotsDonTHaveVisualDifferenceIgnoringLogo()
            {
                _differenceImage = Names.DiffImagePath;
                double                    differencePercent                =
VisualSteps.FindDiffPercentIgnoreElement(_expectedResult,          _actualResult,
_differenceImage, SchedulePage.Logo);
                differencePercent.Should().Be(0);
            }
        }
    }
    using Common.Configuration;
    using Steps.Util;
    using System;
    using System.IO;
```

```
using System.Text.RegularExpressions;
using TechTalk.SpecFlow;

namespace Tests
{
   public static class Names
   {
      public static string TestId => ScenarioContext.Current.ScenarioInfo.Title + DateTime.Now;
      public static string TestTitle => ScenarioContext.Current.ScenarioInfo.Title;

      public static string DiffImagePath => Path.GetFullPath(ScreenshotsDir + DiffImageName);
      public static string ActualImagePath => Path.GetFullPath(ScreenshotsDir + ActualImageName);
      public static string ExpectedImagePath => Path.GetFullPath(@".\TestData\Screenshots\" + ExpectedImageName);
      public static string TestReportPath => Path.GetFullPath(ReportDir + ReportName);

      public static string OutDir => GetOutDir();
      public static string ScreenshotsDir => $@"{OutDir}\Screenshots\";
      public static string ReportDir => $@"{OutDir}\Report\";

      private static string _outDir;
      private static string ActualImageName => GetValidString(TestTitle + "_actual_" + DateTime.Now + ".png");
      private static string ExpectedImageName => GetValidString(TestTitle + "_expected_" + EnumsHelper.GetDescription(CurrentPreferences.screen) + ".png");
      private static string DiffImageName => GetValidString(TestTitle + "_diff_" + DateTime.Now + ".png");
      private static string ReportName => GetValidString("TestReport_" + DateTime.Now + ".html");

      private static string GetValidString(string value) => Regex.Replace(value, @"[^a-zA-Z0-9_.]+", "_");

      private static string GetOutDir()
      {
         if (_outDir == null)
            _outDir = $@".\{Environment.UserName}" + GetValidString(DateTime.Now.ToString());
         return _outDir;
```

```
        }
    }
}
using Common.Configuration;
using Steps.DriverSteps;
using Steps.Util;
using Steps.Util.DB;
using Steps.Util.DB.Models;
using System;
using System.Diagnostics;
using System.IO;
using TechTalk.SpecFlow;

namespace Tests
{
    [Binding]
    public class FeatureActions : VisualTestsBase
    {
        private static DriverHelper webDriver;
        private static Stopwatch testRunWatch;
        private static Scenario Scenario;
        private static Feature Feature;
        private static ResultStatus ResultStatus;
        private static ExecutedTest ExecutedTest;
        private static Screen Screen;
        private static Browser Browser;

        [BeforeTestRun(Order = 0)]
        public static void BeforeTestRunCreateDir()
        {
            if (!Directory.Exists(Names.OutDir))
            {
                Directory.CreateDirectory(Names.ScreenshotsDir);
                Directory.CreateDirectory(Names.ReportDir);
            }
        }

        [BeforeTestRun(Order = 1)]
        public static void BeforeAddReportContent()
        {
            TestReportHelper.CreateStructure();
        }

        [BeforeTestRun(Order = 2)]
        public static void BeforeTestRunStartWatch()
```

```
        {
            testRunWatch = new Stopwatch();
            testRunWatch.Start();
        }

        [AfterTestRun(Order = 0)]
        public static void AfterTestRunStopWatch()
        {
            testRunWatch.Stop();
            TestReportHelper.TotalTime = testRunWatch.Elapsed;
        }

        [AfterTestRun(Order = 1)]
        public static void AfterTestRunWriteToReport()
        {
            TestReportHelper.AddSummary();
            TestReportHelper.WriteToReport(Names.TestReportPath);
        }

        [BeforeFeature(Order = 0)]
        public static void BeforeFeature()
        {
            webDriver = DriverHelper.Instance();
            webDriver.Init(CurrentPreferences.driver,
CurrentPreferences.screen);
        }

        [AfterFeature(Order = 0)]
        public static void AfterFeature()
        {
            webDriver.Dispose();
        }

        [BeforeScenario(Order = 0)]
        public static void BeforeScenarioStartWatch()
        {
            Feature = new Feature();
            Scenario = new Scenario();
            ResultStatus = new ResultStatus();
            Browser = new Browser() { BrowserName =
CurrentPreferences.driver.ToString() };
            Screen = new Screen() { Size =
EnumsHelper.GetDescription(CurrentPreferences.screen) };

            ExecutedTest = new ExecutedTest
```

```
            {
                StartTime = DateTime.Now,
                Browser = Browser,
                Screen = Screen
            };
        }

        [AfterScenario(Order = 0)]
        public static void AfterScenarioStopWatch()
        {
            ExecutedTest.EndTime = DateTime.Now;
        }

        [AfterScenario(Order = 1)]
        public static void AfterScenarioWriteToReport()
        {
            ScenarioContext currentTest = ScenarioContext.Current;
            Feature.FeatureName = FeatureContext.Current.FeatureInfo.Title;
            Scenario.ScenarioName = Names.TestTitle;
            Scenario.Feature = Feature;
            ExecutedTest.Scenario = Scenario;
            ResultStatus.TestStatus =
currentTest.ScenarioExecutionStatus.ToString();
            ExecutedTest.ResultStatus = ResultStatus;

            ExecutedTest.ErrorMessage = currentTest.TestError?.Message;

            TimeSpan duration = ExecutedTest.EndTime -
ExecutedTest.StartTime;

            TestReportHelper.AddTestSummary(Scenario.ScenarioName,
                ResultStatus.TestStatus,
                ExecutedTest.ErrorMessage ?? "-",
                duration,
                _actualResult,
                _expectedResult,
                File.Exists(_differenceImage) ? _differenceImage : string.Empty);
        }

        [AfterScenario(Order = 2)]
        public static void AfterScenarioWriteToTable()
        {
            DBHelper.AddTestResult(Feature, Scenario, Browser, Screen,
ResultStatus, ExecutedTest);
        }
```

```
        }
    }
    using FluentAssertions;
    using Steps;
    using Steps.DriverSteps;
    using TechTalk.SpecFlow;

    namespace Tests
    {
        [Binding]
        public class VisualTestsBase
        {
            protected static readonly DriverHelper Driver =
DriverHelper.Instance();
            protected static readonly VisualSteps VisualSteps = new VisualSteps();
            protected static string _expectedResult => Names.ExpectedImagePath;
            protected static string _actualResult;
            protected static string _differenceImage;

            [When(@"I take page screenshot")]
            public void WhenITakePageScreenshot()
            {
                _actualResult = Names.ActualImagePath;
                Driver.TakePageScreenshot(_actualResult);
            }

            [Then(@"Screenshots don't have visual difference")]
            public void ThenScreenshotsDonTHaveVisualDifference()
            {
                _differenceImage = Names.DiffImagePath;
                double differencePercent =
VisualSteps.FindDiffPercent(_expectedResult, _actualResult, _differenceImage);
                differencePercent.Should().Be(0);
            }
        }
    }
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проєкту	**В.о. завідувача кафедри**
_____ *К. І. Ліщук*	_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)	(підпис) (ініціали, прізвище)
“13” квітня 2020 р.	“14” квітня 2020 р.

Засоби автоматизованого тестування інтерфейсу користувача

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП ІС-6218.01.000 ТЗ*

на 9 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: «Засоби автоматизованого тестування інтерфейсу користувача».

Коротке найменування системи: «UIAutoTestingTools».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником проекту є кафедра автоматизованих систем обробки інформації і управління факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Розробниця застосування – студентка групи ІС-62 факультету інформатики та обчислювальної техніки КПІ ім. Ігоря Сікорського Путова Юлія Андріївна.

1.3 Перелік документів, на підставі яких створюється система

Підставою для розробки “Засоби автоматизованого тестування інтерфейсу користувача” є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням застосування – 13 квітня 2020 року.

Плановий термін по закінченню роботи над створенням застосування – 15 травня 2020 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ ЗАСОБІВ

2.1 Призначення засобів

Розроблювана система призначена для розробки автоматизованих функціональних тестів інтерфейсу користувача та їх виконання. Такі тести є ефективними для тестування веб-застосувань, де зміни в інтерфейсі користувача можуть призвести до втрати коштів замовника, та для таких, де ручне тестування інтерфейсу користувача є менш ефективним та більш часозатратним, ніж пропонований підхід.

2.2 Цілі створення засобів

Ціль розробки – створити фреймворк, що полегшує роботу автоматизаторів тестування за рахунок зменшення часу на розробку та виконання тестів, підвищує ефективність тестів, збільшуючи ймовірність знаходження дефектів (адже при ручному тестуванні дефекти можуть бути просто не помічені), розширює покриття застосування тестами та містить необхідні гнучкі інструменти для ефективного тестування інтерфейсу користувача та виведення результатів тестування у зручному вигляді.

Виконання тестів на різних браузерах та з різними розмірами екрану збільшує покриття програмного продукту тестами, не переписуючи тест кожного разу, що дозволяє якнайшвидше отримати дані про якість продукту.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Тестування програмного забезпечення – це процес аналізу програмного продукту та його документації з метою покращення якості програмного продукту. Тестування дозволяє впевнетись, що програмний продукт відповідає вимогам замовника та визначає умови, при яких робота програми є некоректною.

Об’єктом автоматизації є процес тестування графічного інтерфейсу користувача, а саме елементів вебсторінок, їх розташування, зовнішній вигляд. На етапі тестування інтерфейсу користувача (UI) перевіряється, наскільки він зручний для користувача та чи відповідає він вимогам замовника. При цьому проводиться перевірка того, як відображаються елементи інтерфейсу при виконанні різних дій користувачем.

Одна з задач тестування графічного інтерфейсу – підтвердити, що застосування відповідає макету прототипу. Це перевірка елементів дизайну, таких як колір, шрифт, його розмір, розташування елементів один відносно одного. Таке тестування проводиться після додавання нового функціоналу до будь-якого компоненту або зміни дизайну застосування. Постійне тестування інтерфейсу користувача допомагає виявити дефекти у новому функціоналі або дефекти, що викликані цими змінами.

Тестування UI вебсайтів зазвичай проводиться на різних пристроях та браузерах. Це зумовлено тим, що при різних розмірах екрану пристроїв та у різних браузерах робота та зовнішній вигляд застосувань може відрізнятися. Як правило, вибір комбінацій браузерів, операційних систем та розмірів екрану є задачею замовника.

Головною задачею автоматизації тестування є зменшення часу та зусиль, що потребує ручне тестування.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

У системі, що розробляється, тестувальник програмного забезпечення повинен отримати інструменти для написання сценаріїв, їх імплементації запуску тестів, генерації звіту з виконаних тестів.

Застосування повинно виконувати наступні функції:

– ведення сценаріїв тестування;
– ведення очікуваних результатів для сценаріїв тестування;
– виконання тестових сценаріїв;
– ведення результатів виконання сценаріїв тестування;
– формування звітних форм.

4.2 Вимоги до надійності

4.2.1. Передбачити контроль введення інформації.

4.2.2. Передбачити захист від некоректних дій користувача.

4.2.3. Забезпечити цілісність інформації в базі даних.

4.3 Вимоги до складу і параметрів технічних засобів

Для правильної роботи розробленого застосування до складу технічних засобів повинен входити комп’ютер з операційною системою Windows 7 або вищої версії та зі встановленими браузерами, на яких буде проводитись тестування, та Visual Studio 2017 чи вище. Також комп’ютер має бути підключений до мережі інтернет.

Вимоги до браузерів наведені в таблиці 4.1.

Таблиця 4.1 – Вимоги до браузерів

Найменування браузера	Вимоги до браузера
Google Chrome	Версія не нижче 12.0.712.0
Firefox	Версія не нижче 60
Internet Explorer	Версія 11
Opera	Версія не нижче 62

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки наведені в таблиці 5.1.

Таблиця 5.1 - Основні етапи розробки

№	Назва етапу	Термін виконання
1.	Вивчення рекомендованої літератури	01.03.2020
2.	Аналіз існуючих методів розв’язання задачі	07.03.2020
3.	Постановка та формалізація задачі	15.03.2020
4.	Розробка програмного забезпечення	11.05.2020
5.	Налагодження програми	12.05.2020
6.	Оформлення пояснювальної записки	14.05.2020
7.	Подання ДП на попередній захист	16.05.2020
8.	Подання ДП на основний захист	25.06.2020

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

6.1 Види випробувань

Для перевірки роботи програмного забезпечення буде виконано ряд функціональних випробувань. Ці випробування націлені перевірити правильність роботи компонентів застосування та застосування в цілому. При цьому перевіряється виконання передумов тестових сценаріїв, виконання тестів, порівняння зображень при різних комбінаціях порівняння, вміст згенерованого звіту з виконання тестів при різних результатах виконаних тестів.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
08.06.2020 18:22:35 EEST

Дата звіту:
09.06.2020 02:29:17 EEST

ID перевірки:
1003881983

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: **Putova_bachelor_is62**

ID файлу: **1003896766** Кількість сторінок: **73** Кількість слів: **9973** Кількість символів: **83569** Розмір файлу: **1.11 MB**

4.99% Схожість

Найбільша схожість: **1.01%** з джерело бібліотеки. ID файлу: **5839535**

2.51% Схожість з Інтернет джерелами	37	Page 75
4.21% Текстові збіги по Бібліотеці акаунту	188	Page 76

0.1% Цитат

Цитати	2	Page 77

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	5

Графічний матеріал
до дипломного проєкту

на тему: «*Засоби автоматизованого тестування інтерфейсу користувача*»

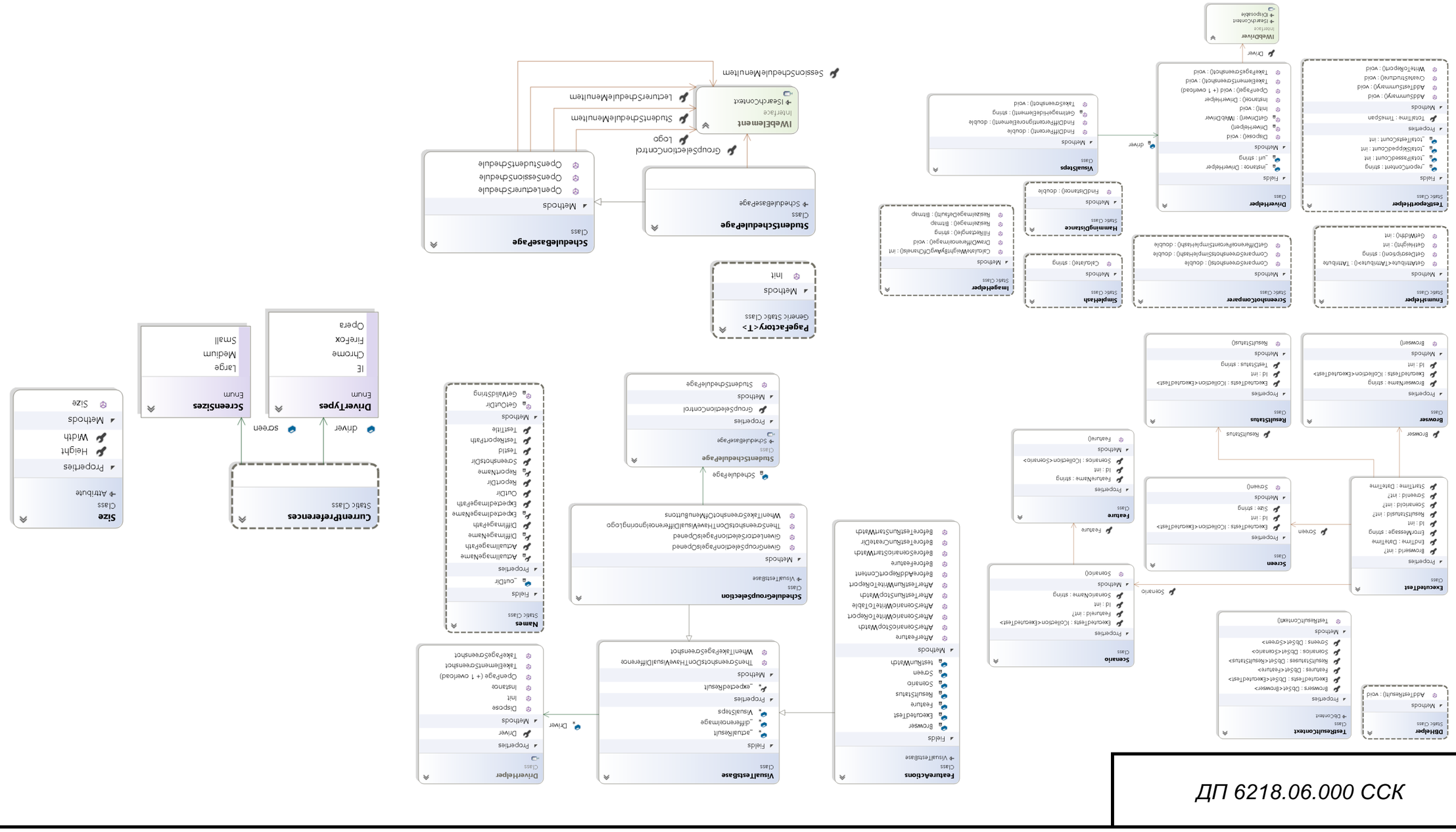
Київ – 2020 року

					ДП 6218.02.000 ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Путова Ю.А.					
Перевірив		Ліщук К.І.				Аркуш 2	Аркушів 9
Т. кон.					Засоби автоматизованого тестування інтерфейсу користувача	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62	
Н. кон.		Новінський В.П.					
Затвердив		Ліщук К.І.					

					ДП 6218.03.000 ССВ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера	Маса / Масштаб
Розробив		Путова Ю.А.					
Перевірив		Ліщук К.І.				Аркуш 3	Аркушів 9
Т. кон.					Засоби автоматизованого тестування інтерфейсу користувача	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62	
Н. кон.		Новінський В.П.					
Затвердив		Ліщук К.І.					

					ДП 6218.04.000 СБД		
					Схема бази даних	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Путова Ю.А.				Аркуш 3	Аркушів 9
Перевірив		Ліщук К.І.					
Т. кон.					*Засоби автоматизованого тестування інтерфейсу користувача*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62*	
Н. кон.		Новінський В.П.					
Затвердив		Ліщук К.І.					

					ДП 6218.05.000 ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Путова Ю.А.					
Перевірив		Ліщук К.І.				Аркуш 5	Аркушів 9
Т. кон.					Засоби автоматизованого тестування інтерфейсу користувача	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62	
Н. кон.		Новінський В.П.					
Затвердив		Ліщук К.І.					



Зм.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Гумова Ю.А.			
Перев.		Лілук К.І.			
Т. Кон.					
Н. Кон.		Новіковський В.Л.			
Затв.		Лілук К.І.			

ДП 6218.06.000 ССК

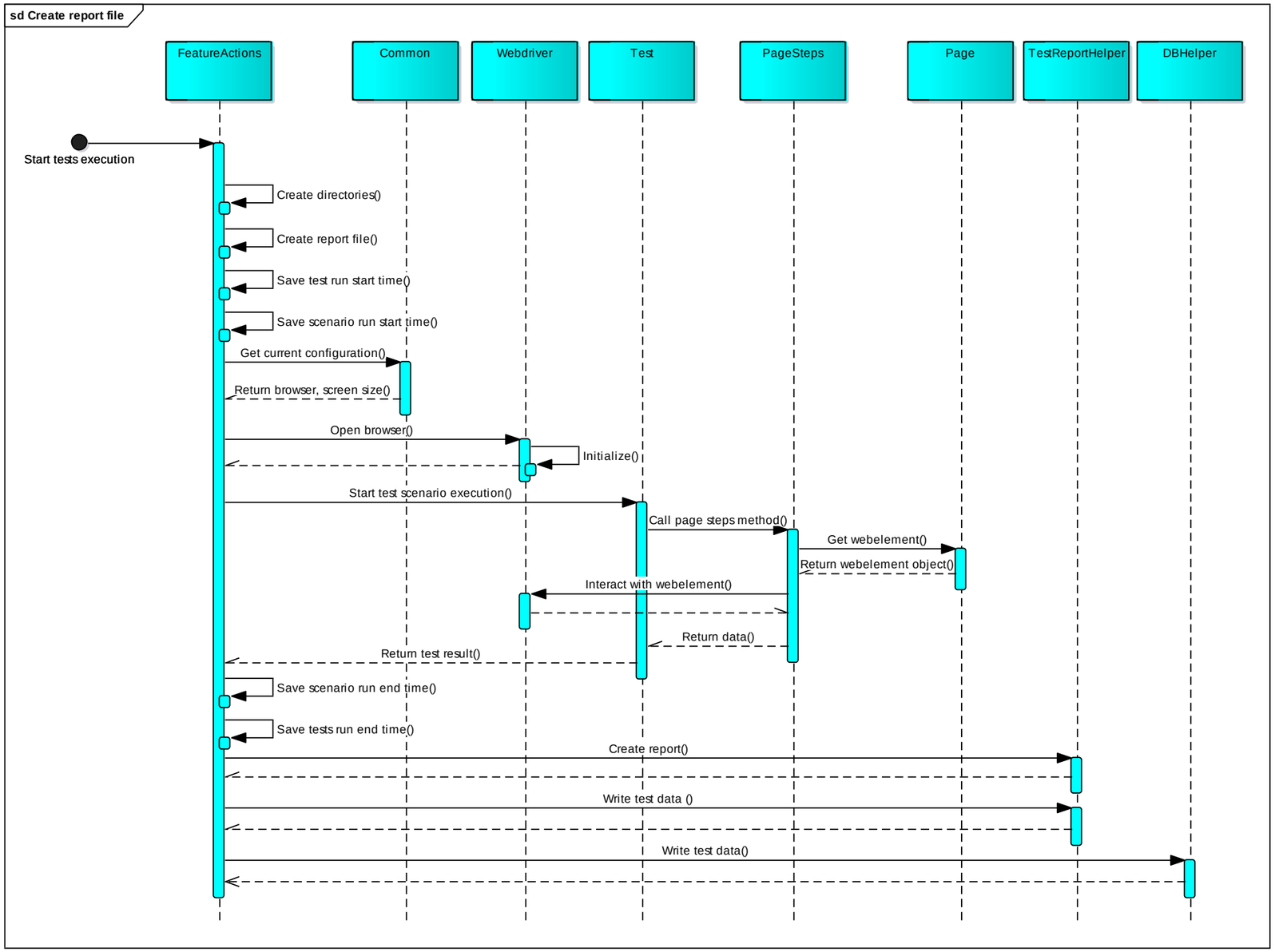
Літера	Маса	Масштаб	
Аркуш 6		Аркушів 9	

Схема структурна класів програмного забезпечення

Засоби автоматизованого тестування інтерфейсу користувача

КПІ ім. Ігоря Сікорського
кафедра АСОІУ гр. ІС-62

ДП 6218.07.000 ССП



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Путова Ю.А.		
Перев.		Ліщук К.І.		
Т. Кон.				
Н. Кон.		Новінський В.П.		
Затв.		Ліщук К.І.		

ДП 6218.07.000 ССП

Схема структурна послідовності

Літера	Маса	Масштаб

Аркуш 7	Аркушів 9

Засоби автоматизованого тестування інтерфейсу користувача

КПІ ім. Ігоря Сікорського
кафедра АСОІУ гр. ІС-62

ДП 6218.08.000 КЕ

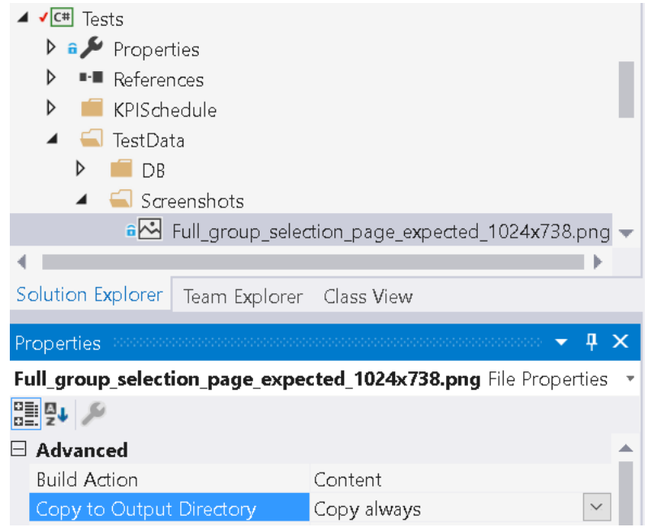
ScheduleGroupSelection.feature

```
Feature: ScheduleGroupSelection

Scenario: Full group selection page
    Given gr
    When I t
    Then Scr
```


ScheduleGroupSelectionSteps.cs

Miscellaneous Files | Tests.ScheduleGroupSelection

```
using FluentAssertions;
using Pages;
using Steps;
using TechTalk.SpecFlow;
namespace Tests
{
    [Binding, Scope(Feature = "ScheduleGroupSelection")]
    public class ScheduleGroupSelection : VisualTestsBase
    {
        private static StudentSchedulePage SchedulePage;

        [Given(@"group selection page is opened")]
        public void GivenGroupSelectionPageIsOpened()
        {
            Driver.OpenPage();
            SchedulePage = new StudentSchedulePage();
            PageFactory<StudentSchedulePage>.Init(SchedulePage, Driver.Driver);
            SchedulePage.OpenStudentSchedule();
        }
```



ScreenSizes.cs

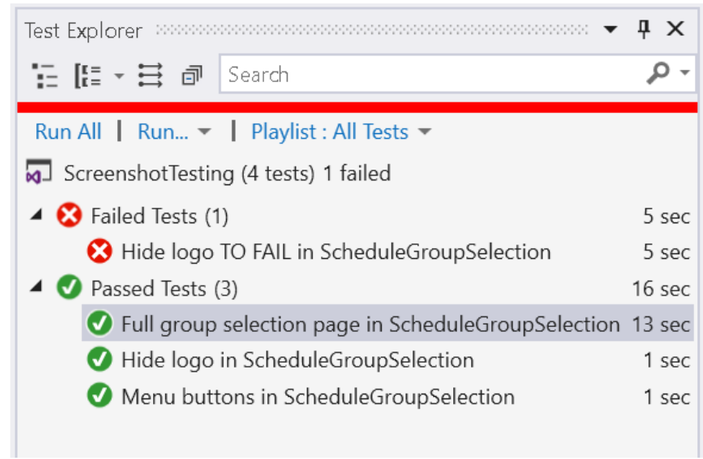
Common | Common.Enums.Size

```
using System;
using System.ComponentModel;

namespace Common.Enums
{
    public enum ScreenSizes
    {
        [Description("1366x768")]
        [Size(1366, 768)]
        Large,

        [Description("1280x800")]
        [Size(1280, 738)]
        Medium,

        [Description("1024x738")]
        [Size(1024, 738)]
        Small
    }
```

CurrentPreferences.cs

Common | Common.Configuration.CurrentPreferences

```
using Common.Enums;

namespace Common.Configuration
{
    public static class CurrentPreferences
    {
        public static DriverTypes driver = DriverTypes.Chrome;
        public static ScreenSizes screen = ScreenSizes.Medium;
    }
}
```



ScreenshotTesting > TestResults > ya31.05.2020_12_04_00 > Report

Имя	Дата изменения	Тип	Размер
TestReport_31.05.2020_12_04_22.html	31.05.2020 12:04	Chrome HTML Docu...	4 КБ

ScreenshotTesting > TestResults > ya31.05.2020_12_04_00 > Screenshots


Full_group_selection_page_actual_5_31_2020_12_04_08_PM.png


Hide_logo_TO_FAIL_actual_5_31_2020_12_04_18_PM.png


Hide_logo_TO_FAIL_diff_5_31_2020_12_04_18_PM.png

Зм.	Арк.	№ докум.	Підпис	Дата	ДП 6218.08.000 КЕ		
Розроб.		Путова Ю.А.			Креслення вигляду екранних форм	Літера	Маса / Масштаб
Перев.		Ліщук К.І.					
Т. Кон.						Аркуш 8	Аркушів 9
Н. Кон.		Новінський В.П.			Засоби автоматизованого тестування інтерфейсу користувача	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62	
Затв.		Ліщук К.І.					

ДП 6218.09.000 КЗ

Test Report from 31.05.2020 12:04:01

Chrome, 1280x800

Summary

Tests run	4
Passed	3
Failed	1
Skipped	0
Success rate	75%
Total time	00:00:21.3928771

Tests

1. Full group selection page

Result	Error	Duration
OK	-	00:00:02.0107903

▶ Screenshots

2. Menu buttons

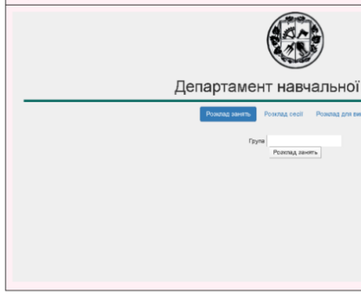
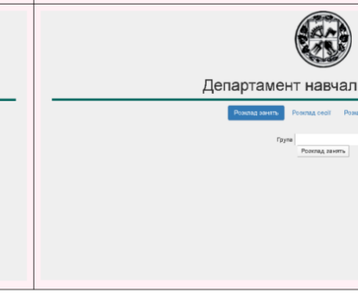
Result	Error	Duration
OK	-	00:00:01.1868272

▶ Screenshots

3. Hide logo

Result	Error	Duration
OK	-	00:00:01.1413902

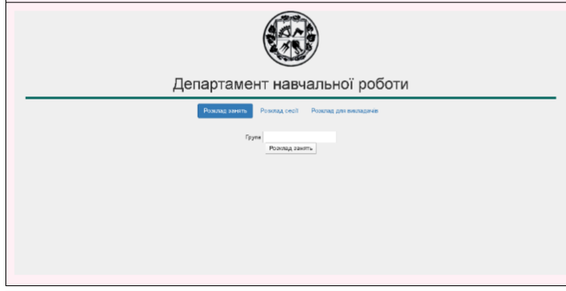
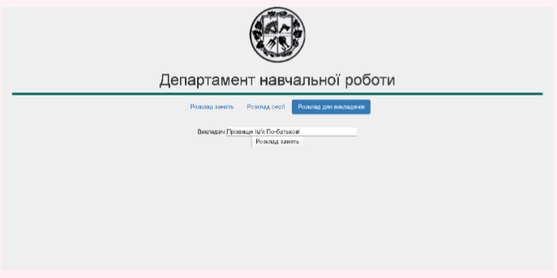
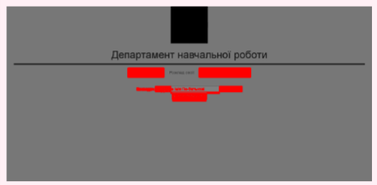
▼ Screenshots

Expected	Actual	Difference
		

4. Hide logo TO FAIL

Result	Error	Duration
TestError	Expected differencePercent to be 0.0, but found 2.63671875.	00:00:03.9177997

▼ Screenshots

Expected	Actual	Difference
		

Зм.	Арк.	№ докум.	Підпис	Дата	ДП 6218.09.000 КЗ		
Розроб.		Путова Ю.А.			Креслення вигляду звітних форм	Літера	Маса / Масштаб
Перев.		Ліщук К.І.				Аркуш 9	Аркушів 9
Т. Кон.					Засоби автоматизованого тестування інтерфейсу користувача	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-62	
Н. Кон.		Новінський В.П.					
Затв.		Ліщук К.І.					